VLSI ARCHITECTURES FOR MIMO DETECTION

By

RAMIN SHARIAT-YAZDI, B.Sc., M.A.Sc.

A thesis submitted to the faculty of graduate studies and research in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Electronics
Carleton University
Ottawa, Ontario
January 2010

Your file Votre référence
ISBN: 978-0-494-63865-1
Our file Notre référence
ISBN: 978-0-494-63865-1

The undersigned recommend to the Faculty of Graduate Studies and Research, the acceptance of the Ph.D. thesis:

"VLSI architectures for MIMO detection"

Submitted by

Ramin Shariat-Yazdi, B.Sc., M.A.Sc.

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Professor Q. J. Zhang, Chair, Department of Electronics

Professor Tad Kwasniewski, Thesis Supervisor

Dr. Valek Szwarc, External Examiner

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Electronics

Carleton University

January 2010

Important Information

The information used in this thesis comes in part from the research program of Dr. Tad A. Kwasniewski and his associates in the VLSI in Communications group. The research results appearing in this thesis represent an integral part of the ongoing research program. All research results in this thesis including tables, graphs, and figures but excluding the narrative portions of the thesis are effectively incorporated into the research program and can be used by Dr. Kwasniewski and his associates for education and research purposes, including publication in open literature with the appropriate credits. Matters of intellectual property may be pursued cooperatively with Carleton University and Dr. Kwasniewski where and as applicable.

Abstract

Multiple-Input Multiple-Output (MIMO) communication is a rapidly developing wireless technology that promises wireless access with unprecedented data rates to multiple users at high quality of service (QoS). These benefits stems from employing multiple antennas on both sides of the wireless channel and transmitting multiple data streams concurrently and in the same frequency bands, implying higher channel capacity and link reliability. One of the main challenges in practical realization of MIMO wireless systems lies in the efficient implementation of the MIMO detectors.

This dissertation covers design and implementation of high performance MIMO detectors. In particular tree search based detection algorithms are studied as the basis for developing new detection algorithms and architectures. A new approach has been proposed for design of MIMO detectors based on a configurable processing element that can be utilized in different configurations for implementation of tree search based detection algorithms. Based on a novel enumeration algorithm for M-QAM constellation space, a low complexity dynamically configurable processing element (*MECU*) that can be used as the main processing element for implementation of various MIMO detectors has been designed. The *MECU* processing element can be dynamically configured to support BPSK, QPSK, 16-QAM and 64-QAM modulation and is characterized by a low silicon complexity equivalent to 18K gates.

Depth-first-K-Best (*DFKB*) sphere detection algorithm is proposed to overcome major short comings of the classical depth-first sphere decoding algorithm in throughput performance and memory space requirements. A folded architecture implementation of this algorithm based on *MECU* processing element achieves a maximum throughput of 672 Mbps and supports combinations of antenna configurations (4x4, 3x3, 2x2) over a configurable range of modulation modes (BPSK, QPSK, 16-QAM and 64-QAM).

Fixed sphere decoding algorithm is a sub-optimal fixed throughput detection algorithm that can be implemented in hardware using an array of *MECU* processing elements. An array processor for a 4x4 (8-1-1-1) decoder based on *MECU* is able to achieve a maximum throughput of 3600 Mbps.

In case of breadth-first detectors, it is shown that by applying 2-stage node elimination algorithm instead of the strict K best search, the overall algorithmic and architectural complexity of the complex-valued K-best detector can be reduced significantly. A complex-valued pipelined VLSI architecture for K-best algorithm using *MECU* processing elements is designed and implemented. The implementation results for a 4x4 (QPSK, 16-QAM and 64-QAM) detector compared to real-valued reference detector shows 53% decrease in silicon complexity and 75% increase in maximum clock frequency. Implementation of the proposed complex-valued K-best detector in CMOS 0.13 μm technology achieves up to 1680 Mbps throughput at 107K gates.

In order to demonstrate and evaluate performance of *DFKB-MECU* architecture, a configurable 4x4/3x3/2x2 64-QAM/16-QAM/QPSK/BPSK ASIC detector has been designed and routed in CMOS 0.18 μm technology. The core area of the detector is 0.87mm^2 and it can achieve a throughput of 288 Mbps.

Acknowledgments

I take this opportunity to thank my supervisor, Professor Kwasniewski for his guidance, support and the freedom to choose and pursue the topic of my interest.

I thank Professor Calvin Plett, Professor Ralph Mason, Professor Voicu Groza and Dr. Valek Szwarc members of my defense committee for taking the time to review this manuscript and providing valuable comments for improving the quality of this dissertation.

I have been very fortunate to work with many graduate students in the VLSI signal and data processing group. I would like to specially thank Irfan Ahmed and Peter Noel for valuable discussions on various topics over the past few years.

I appreciate the support from the staff of the Electronics Department who supported me over the course of my studies. My special thanks go to Nagui Mikhail, Blazenka Power, Scott Brue, Anna Lee and Jacques Lemieux.

I show my gratitude to my parents for their continuous support, love, and encouragements over the course of my studies. My late father passed away one week before my final defense and I always remember him for being there for me whenever I needed him. For more than words can express I show my gratitude to my mother for her care, love and believing in me.

Our beautiful daughter Yasmine has been the source of joy in our lives and kept my spirit going over the past three years. She was always there to make me smile and forget about everything else.

My final, and most heartfelt, acknowledgment goes to my wife Maryam for always standing beside me throughout my studies. Her continuous love, support, encouragement and dedication made this work possible. I thank her for her patience and sacrifices.

Dedication

To the memory of my late father, my mother and the love of my life Maryam

Table of Contents

List of Figures

List of Tables

List of Symbols

N_t .....number of transmitter antennas

N_r .....number of receiver antennas

$\mathbf{y}$.....$N_r x1$ vector of received symbols

$\mathbf{H}$ .....$N_r \times N_t$ equivalent flat-fading channel matrix

$\tilde{\mathbf{H}}$ .....effective channel matrix

$\mathbf{G}$ .....estimator matrix in linear detection

$\mathbf{Q}$ .....orthogonal matrix in QR decomposition

$\mathbf{R}$.....upper triangular matrix in QR decomposition

$\mathbf{s}$ .....$N_t x1$ vector of transmitted symbols.

$\hat{\mathbf{s}}$ .....linear estimation of vector $\mathbf{s}$

$\mathbf{n}$ .....$N_r x1$ vector of Gaussian thermal noise

$\mathbf{n_{ZF}}$..... zero forcing estimator equivalent thermal noise

$\mathbf{n_{MMSE}}$..... MMSE estimator equivalent thermal noise

σ^2.....variance

L_A .....a priori L-value

L_E .....extrinsic L-value

L_D .....a posteriori L-value

f_{clk}.....operating clock frequency

R.....uncoded transmission rate

Λ.....complex constellation space

Λ_R.....transformed Λ constellation space

Q_b.....number of bits per symbol in Λ

P_c.....number of complex valued constellation points in Λ

P_r.....number of real valued constellation points in Λ

$p(\mathbf{y} \mid \mathbf{x})$ .....likelihood function

D_{avg} .....average processing cycles

f_{clk}.....operating clock frequency

$T_i(\mathbf{P}_i)$ .....node metric - Partial Euclidean Distance (PED)

$e_i(\mathbf{P}_i)$ .....branch metric-metric increment

$\mathbf{P}_i$ .....partial symbol vector

r .....sphere radius

L .....candidate list in LSD algorithm

$\mathbf{s}_{ML}$ .....maximum likelihood solution

$16\{0xN_3N_2N_1N_0\}$ .....16-bit hexadecimal number

p_r .....real partition code

p_i .....imaginary partition code

$\mathbf{s}_{0r}$ .....real constellation code

$\mathbf{s}_{0i}$ .....imaginary constellation code

$P_{dynamic}$ .....dynamic power dissipation

C_L .....node capacitive load

α .....activity factor

E_b/N_o .....energy per bit to noise spectral density

N_t .....Number of transmitter antennas

N_r .....Number of receiver antennas

DR .....Data rate

c_m .....constellation point in M-QAM space Λ

$\mathbf{P}_i(c_m)$ .....Path over the search tree from root to node c_m

$T_i(c_m)$ .....PED metric associated with $\mathbf{P}_i(c_m)$

List of Abbreviations and Acronyms

ALM.....Adaptive Logic Module
ALUT.....Adaptive Look-Up Table
APP.....A Posterior Probability
ASIC.....Application Specific Integrated Circuit
BER.....Bit Error Rate (Ratio)
BFTS.....Breadth-First Tree Search
bpcu..... bit per channel use
BPSK.....Binary Phase Shift Keying
CMC.....Canadian Microsystems Corporation
CMOS.....Complementary Metal-Oxide Semiconductor
cmpx.....compare and exchange cell
DDC.....Digital Down Conversion
DFKB.....Depth-First-K-Best
DFSD.....Depth-First Sphere Decoding
DFTS.....Depth-First Tree Search
DRC.....Design Rule Check
DSP.....Digital Signal Processor/Processing
DUC.....Digital Up Conversion
FFT.....Fast Fourier Transform
FPGA.....Field Programmable Gate Array
FOE.....Frequency Offset Estimation
FOC.....Frequency Offset Compensation
FOM.....Figure OF Merit
FR.....Feedback Register
FSD.....Frame Start Detection
FSD.....Fixed Sphere Decoding
GI.....Guard Insertion

GOPS .....Giga Operations Per Second

GR .....Guard Removal

HDL .....Hardware Descriptive Language

HIL .....Hardware In the Loop

IFFT .....Inverse Fast Fourier Transform

i.i.d .....independent identical distributed

KB .....K-Best

KSE .....K-best Schnorr-Euchner

LAB .....Logic Array Block

LBSR .....Level Based Sphere Radius

LCCE .....Low Complexity Cartesian Enumeration

LDPC .....Low-Density Parity Check

LE .....Logic Element

LSD .....List Sphere Decoder

LUT .....Look-Up table

LVS .....Layout VS Schematic

MCU .....Metric computation Unit

MEU .....Metric Enumeration Unit

MECU .....Metric Enumeration and Computation Unit

MIMO .....Multiple-Input Multiple-Output

MIPS .....Million Instructions Per Second

ML .....Maximum Likelihood

MOPS .....Million Operations Per Second

MMSE .....Minimum Mean Square Error

MPEG .....Moving Picture Expert Group

MPEG-2 .....Moving pictures and audio information coding

NoC .....Network On Chip

OFDM .....Orthogonal Frequency Division Multiplexing

OSIC .....Ordered Successive Interference Cancellation

PE .....Processing Element

PED .....Partial Euclidean Distance

PDFKB.....Parallel Depth First K-Best

PHB.....Path History Buffer

PR.....Pipeline Register

QAM.....Quadrature Amplitude Modulation

QPSK.....Quadrature Phase Shift Keying

RF.....Radio Frequency

RISC.....Reduced Instruction Set Computer

RTL.....Register Transfer Level

RVD.....Real-Valued Decomposition

SC.....Sphere Constraint

SD.....Sphere Decoder (Detector)

SE.....Schnorr-Euchner

SR.....Sphere Radius

SIC.....Successive Interference Cancellation

SISO.....Single Input Single Output

SNR.....Signal to Noise Ratio

TSMC.....Taiwan Semiconductor Manufacturing Company

V-BLAST.....Vertical Bell Laboratories Layered Space Time

WLAN.....Wireless Local Area Network

WMAN.....Wireless Metropolitan Area Network

ZF.....Zero Forcing

Chapter 1 Introduction

1.1 Introduction

Over the passed two decades wireless communication has emerged as one of the fastest growing sectors in end user consumer market. In 1990, there were close to 10 million cell phone subscribers worldwide. In less than twenty years in early 2009, the total number of cell phone subscribers in the world has been estimated to be 4.1 billion. This trend is expected to continue and by 2012, the total number of global cell phone subscribers reaches 4.5 billion. The wireless local area network (WLAN) and wireless metropolitan area network (WMAN) markets also witness a double digit growth in the same period.

The future wireless market growth is mainly driven by the data centric applications rather than the voice centric usage of the early cell phones which was the main market for wireless products in the 90's and early 2000. The evolution in new standards and technologies follows this market trend by introducing communication technologies that satisfy the requirements for higher quality of service (QoS) and higher data rates [37].

Wireless communication systems are limited by the availability of frequency spectrum and as a result expansion of used bandwidth as a solution to channel capacity increase problem has always been problematic. The major challenge facing the future wireless communication networks is to provide high data rate wireless access to multiple mobile users at high quality of service without major increase in the utilized frequency spectrum. Multiple-input multiple-output (MIMO) communication offers a new approach to communication channel capacity increase problem by improving spectral efficiency through spatial multiplexing and use of multiple antennas on both transmitter and receiver. The link reliability is also improved in MIMO by applying antenna diversity and space-time coding techniques [39]. Multiple-input multiple-output (MIMO) technology is considered to be the most significant advancement and the major driver behind the technological achievements of wireless industry [60].

MIMO is the key technology in achieving higher throughput in the next generation wireless standards such as IEEE 802.11n, IEEE 802.16 (WIMAX) as well as 3G and post-3G cellular system. The IEEE 802.11n (WLAN) and IEEE 802.16 (WMAN)

standards define wireless transmission based on the combination of MIMO with orthogonal frequency division multiplexing (OFDM) modulation or MIMO-OFDM [31]. In mobile access technology, there has been an effort by the 3rd Generation Partnership Project (3GPP release 6) to introduce antenna array technologies such as beam forming and MIMO into the second phase of High-Speed Downlink Packet Access (HSDPA) specification [32]. The future 3GPP roadmap after HSDPA is being developed in Long Term Evolution (LTE) project, which aims at up to 100 Mbps data rate for downlink and 50 Mbps for uplink. As in WLAN/WMAN networks, for MIMO based systems, orthogonal frequency division multiplexing (OFDM) appears to be more favourable than code division multiple access (CDMA) as the choice for modulation technique in 4G networks.

The improvements in channel capacity and link reliability provided by MIMO comes at a significant increase in complexity of signal processing algorithms of both transmitter and receiver blocks compared to the existing single antenna transceivers. Such improvements can be achieved by applying complex signal processing algorithms and novel hardware architectures for implementations of these algorithms considering various challenges that exist in design of hardware systems for wireless devices [38].

1.2 Focus of the dissertation and motivation

In a MIMO communication system comprising of N_t transmitter and N_r receiver antennas, each of the N_r receiver antennas receives transmitted signal components consisting of line-of-sight and reflected signal arrays from each of the N_t transmitters. Figure 1.1 depicts a typical MIMO system. The serial transmitted data, after coding and modulation are mapped to N_t transmitting channels and are transmitted from the N_t transmitting antennas. The received signals on each of the N_r receiver antennas include the line of sight signals as well as the multi-path reflected signal components causing destructive MIMO channel effects. In order to recover the original transmitted symbols, the receiver performs combined detection, demodulation and decoding operations on the received signals from the N_r receiver antennas.

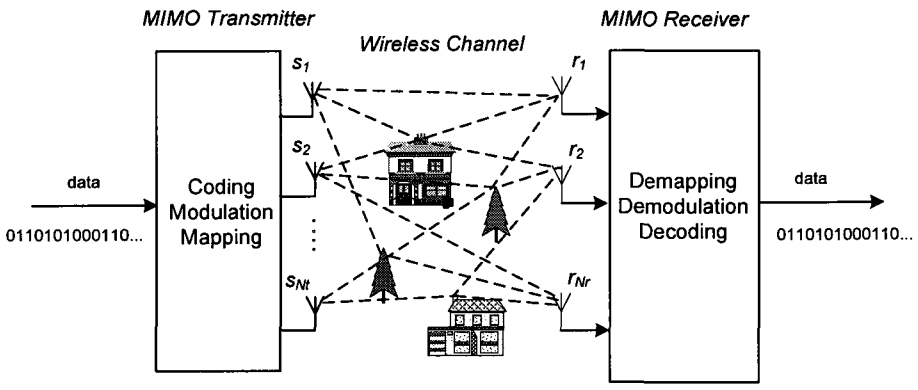


Figure 1.1 MIMO channel

The core idea in MIMO is to use multipath propagations as a resource to increase data rate. The drawback to this performance improvement is considerable increase in complexity of signal processing algorithms needed for separation of parallel data stream in the receiver block. A major challenge associated with the implementation of MIMO communication systems is the design of low complexity MIMO detection algorithms and corresponding VLSI architectures. MIMO detectors are used in receiver blocks in order to recover transmitted symbols from the received symbol vectors.

The objective of this dissertation is to propose low complexity high performance configurable VLSI architectures for hard decision MIMO detectors. For this purpose new algorithms have been proposed and optimized for hardware implementation and dedicated VLSI architectures are proposed for implementation of those algorithms. An algorithmic study provides the opportunity to optimize hardware architecture, gain lower algorithmic complexity and improve performance (bit-error-rate and data throughput). An architectural study on the other hand improves performance, silicon complexity and power consumption.

Figure 1.2 shows an overall overview of MIMO detection algorithms. The gray shaded blocks show the scope of this dissertation from algorithmic perspective. The focus of this thesis will be mainly on tree search based algorithms and dedicated VLSI architectures for implementation of this group of algorithms. The major challenges are maximizing throughput, minimizing silicon complexity and designing dynamically configurable hardware architectures that can support multiple operating modes. Tree search algorithms are also used in searcher block of soft-decision MIMO detectors and efficient

implementation of these algorithms is the first step in implementation of soft decision MIMO detectors.

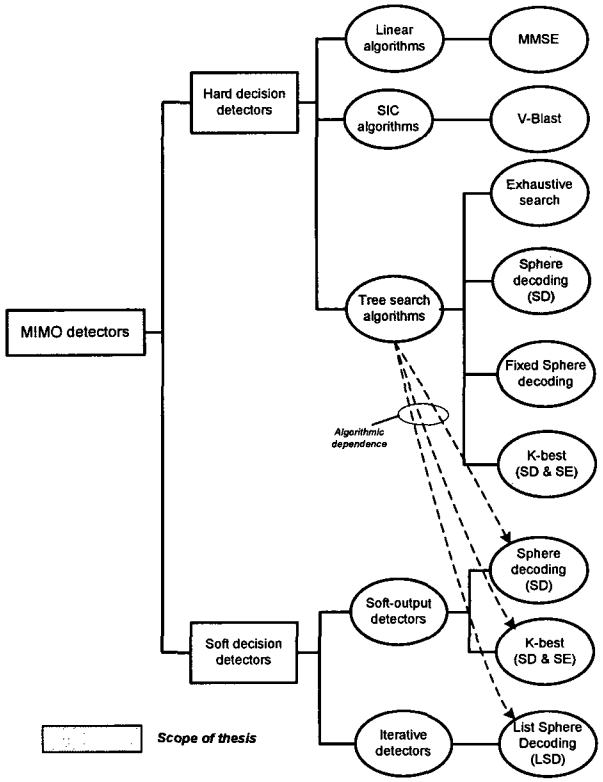


Figure 1.2 MIMO detection algorithms and architectures-classification and thesis scope

1.3 Implementation challenges of MIMO detectors

One of the most challenging components in a MIMO receiver in terms of complexity of algorithm is the MIMO detector. In this section some of the challenges of implementing a high performance MIMO detector will be reviewed. We will also discuss the shortcomings of different implementation platforms in addressing those challenges.

1.3.1 Silicon Area

The performance gains achievable in MIMO-OFDM systems come at a significant increase in silicon area of both transmitter and receiver circuits; as an example the ASIC presented in [42] contains the baseband digital signal processing functional blocks of the physical layer of a MIMO receiver, including an MMSE ordered-successive interference-cancellation (OSIC) MIMO detector. The die area breakdown of the receiver ASIC based

on functional blocks along with die area figures for a corresponding single antenna (SISO) system is summarized in Table 1.1.

From these data it can be observed that compared to a single antenna transceiver, the 4×4 MIMO transceiver requires the four-fold replication of most of the functional blocks in addition to a channel-matrix preprocessor for MIMO detection and the MIMO detector itself. On average the overall chip area in a MIMO receiver increases by a factor of 6.5. As can be seen in this table, the total area associated with the MIMO detector is close to 26% of the total area of the MIMO receiver thus optimizing the area of detector will have a major effect on the overall receiver silicon area.

Table 1.1 Implemented silicon area of baseband functional blocks [42]

Component	Area (mm^2)		
	Single antenna	*4x4 MIMO*	
DDC[1], DUC[2]	0.5	1.9	
AGC[3]	0.1	0.4	
FOE[4], FOC[5], FSD[6]	0.3	1.3	
Modulator, IFFT	0.9	1.4	
Frame buffers	-	3.3	
Channel estimation	0.1	1.12	
QR decomposition	-	1.29	**MIMO detector area = 3.42 ≈ 26% total Area**
QR memory	-	1.23	
MIMO detector	-	0.9	
Receiver total area	**1.9**	**12.8**	

1.3.2 Throughput performance

The main goal in MIMO communication is to increase the overall data throughput of the communication system. One of the limiting factors in achieving this goal is the processing speed of different processing platforms represented by the number of operations per second that can be executed by the processor. There are three main implementation platforms for implementing MIMO detection algorithms; i.e. programmable microprocessors, programmable DSPs and dedicated hardware implementations (ASICs or FPGAs).

[1] Digital Down Conversion
[2] Digital Up Conversion
[3] Automatic Gain Control
[4] Frequency Offset Estimation
[5] Frequency Offset Compensation
[6] Frame Start Detection

Figure 1.3 shows the range of computation performance of each of these platforms [47]. The processing power requirements of some of the previously implemented MIMO detection algorithms are listed in Table 1.2. These numbers show that programmable DSPs and microprocessors cannot meet throughput requirements of MIMO detectors and dedicated VLSI architectures remain as the only viable choice for implementation of detection algorithms.

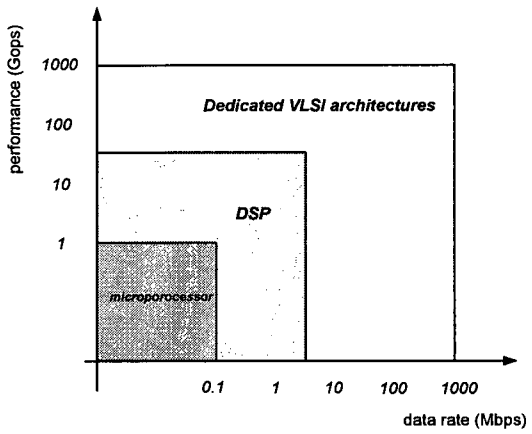


Figure 1.3 Computation performance-source [47]

Table 1.2 Required computation power for MIMO algorithms [47]

	4x4 QPSK ML Detector 28 Mbps	6x6 QPSK ML Detector 42 Mbps	4x4 16-QAM MMSE Detector 100 Mbps	4x4 16-QAM Sphere Detector 200 Mbps
Required *GOPS*	448	588	10	200

1.3.3 Power consumption

There is a strong demand for higher level of processing power which is mainly driven by the high demand signal processing applications. Power consumption will be the limiting factor in reaching this goal. By comparing the power consumption in Intel processors in recent years we can observe an increase of more than 2% in power consumption for every 1% increase in microprocessor performance, which results in a poor MOPS/mW ratio [52]. The same argument is valid for DSPs processors.

Figure 1.4 shows MOPS/mW ratio for different signal processing applications. The processing power demand of a number of data processing algorithms implemented using dedicated hardware processors has also been depicted in Figure 1.4 for comparison to the

average performance of programmable processors and DSPs. As can be seen, dedicated hardware processor implementations are on average more than 100 times more energy efficient than general purpose microprocessors and 10 times more efficient than programmable DSPs. In mobile devices the power efficiency of the signal processor described in MOPS/mW or Mbps/mW ratios are important design factors, dedicated hardware processors can be a viable choice for implementation of various high processing power demand algorithms.

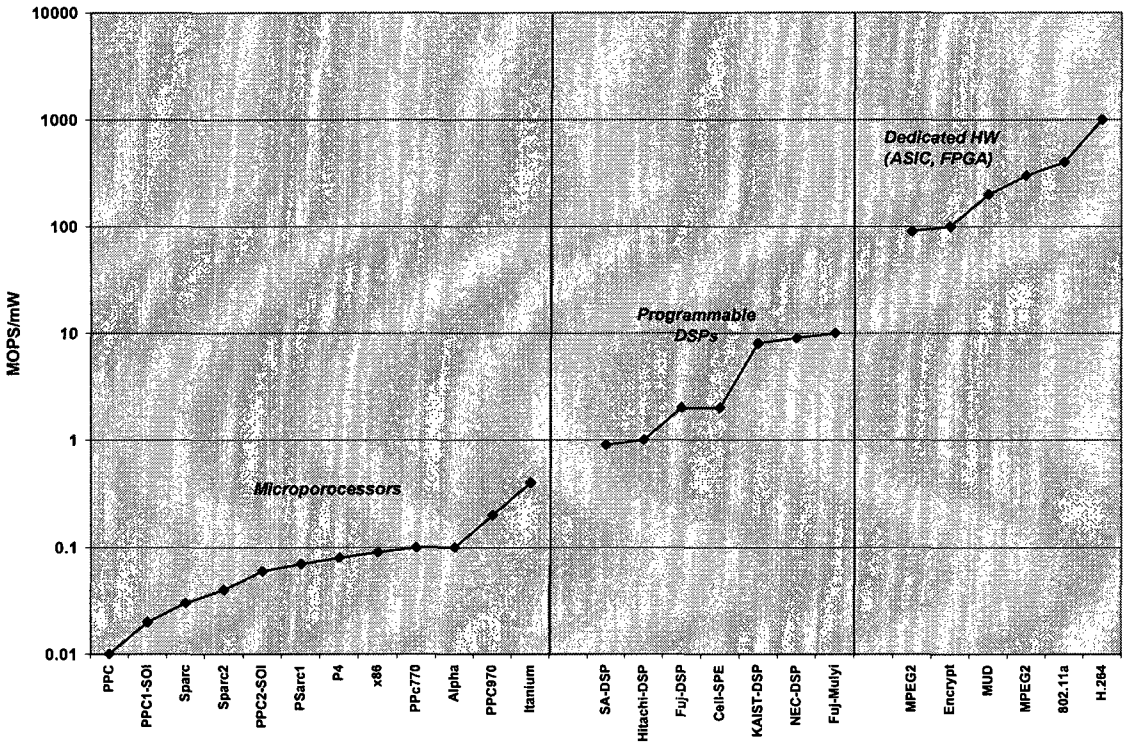


Figure 1.4 Comparison of MOPS/mW for different DSP applications- source [52]

1.3.4 Hardware flexibility

Next generation wireless receivers should be able to support multiple operating modes. These operating modes include programmable number of transmitter/receiver antenna configurations, modulation schemes and coding rates. As mentioned in sections 1.3.2 and 1.3.3, programmable processors and DSPs are not a viable choice for implementation of MIMO receivers due to their low performance and relative high power consumptions; on the other hand dedicated hardware architectures are not inherently programmable and have low level of flexibility. Reconfigurable hardware architectures like FPGAs,

overcome the performance limitations of DSPs while to some extent maintain the flexibility of a programmable device. However, there is a substantial cost and power dissipation overhead associated with flexibility in these devices and most of the operating mode settings are performed on a run-time basis.

Dedicated ASICs on the other hand provide much lower power consumption and smaller silicon area but they lack the inherent flexibility that exists in software programmable devices and to some extent in FPGAs.

The concept of reconfigurable computing has been around since 1960s; Gerald Estrin's [43] proposed the concept of a computer consisting of a standard processor and an array of reconfigurable hardware. In this architecture, the main processor controls the behaviour of the configurable hardware. This results in a hybrid computer structure combining the flexibility of software with the speed of hardware.

Figure 1.5 shows a configurable architecture for MIMO baseband processing. Due to power and throughput limitations discussed in the previous sections, the baseband processing block consists of a combination of dedicated hardware (HW) blocks and programmable processors. The baseband processor reconfigures dedicated hardware blocks for optimum performance and desired operating modes.

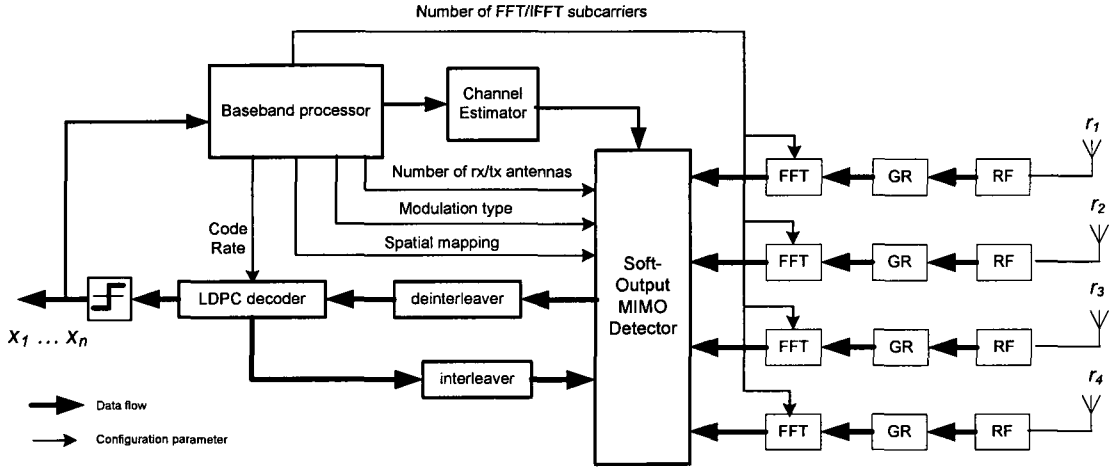


Figure 1.5 Architecture of baseband processing block

1.4 VLSI implementation considerations

1.4.1 Hardware complexity analysis of algorithm

The complexity analysis of MIMO detection algorithms has been mostly based on evaluation of their complexity orders and also estimating number of required adders and multipliers which has been used mainly for qualitative comparison between algorithms. A more detailed complexity analysis and algorithm optimizations for complexity reduction are often performed with digital signal processor (DSP) implementations in mind.

Due to the differences between DSPs and dedicated VLSI circuits, the corresponding complexity estimates do not accurately reflect the true silicon complexity of an optimized VLSI implementation. For the same reason some of the low-complexity algorithms that were optimized for software implementation turned out to be ill suited for ASIC implementations.

The first step toward VLSI implementation of MIMO receivers is to evaluate the hardware resources and operations required by the candidate algorithm for a given set of system parameters (number of antennas, data rate and modulation scheme) in combination with overall system performance (BER and throughput). The following factors should be considered while evaluating an algorithm for hardware implementation:

- The relative VLSI complexity of arithmetic operations. Efficient implementation of arithmetic modules and optimum combination of arithmetic operations can reduce overall complexity of the design.
- Data dependencies determine the degree of dependence between steps in the algorithm, and consequently limit the number of operations that can be completed in parallel to increase throughput.
- Numerical precision is concerned with the number of significant bits in number representation and directly affects the required accuracy of arithmetic operations in an algorithm. Reduced numerical precision and approximations of complex expressions reduce area, but introduce quantization noise. The goal is to reduce complexity with minimal implementation loss of system performance.
- Memory requirements often are associated with increased latency and silicon area.

1.4.2 Mapping algorithms to hardware

The process of mapping an algorithm to hardware starts with architectural exploration. Most of the existing MIMO algorithms are developed with a pure system level perspective in mind and are mostly targeted to microprocessor based platforms. As a result of such an approach these algorithms are not directly implementable in hardware. The following hardware issues are not usually addressed in algorithm development and should be dealt with as part of hardware implementation:

- Pipelining and parallel processing
- Optimum clock frequency
- Implementation of mathematical computation blocks
- Optimum size of data paths
- Floating-point vs. fixed-point data representation
- Architecture of the main control unit that coordinates data flow
- Configurability
- System on chip (SoC) related issues:
    - Timing closure
    - Multiple clock domains
    - Internal buses
    - Memory implementations
    - Scalability of hardware intellectual property (IP) cores

1.4.3 Architectural exploration

Circuit designers trade off power and area for delay (speed) using architectural transformations and optimizations. The optimal VLSI architecture for an algorithm is technology dependant, which requires characterization and optimization of main functional blocks for speed, power and area. Data throughput is one of the main constraints in ASIC design. The basis for circuit optimization is technology specific energy-delay and area-delay trade off curves as show in Figure 1.6. The architecture is power efficient if the desired trade off point in data-path logic is aligned with overall delay [54]. In more complex designs, area or power is considered as the main constraint and the goal is to minimize delay while achieving area or power constraint.

The following should be considered while choosing an architecture:

- Clock frequency selection: The choice of clock frequency will improve the overall speed while increasing the power.
- Parallel processing: replicates processing elements such as arithmetic units to increase throughput at the expense of substantial additional area.
- Iterative decomposition and time sharing: schedule multiple operations to a single resource to save area at the expense of throughput.
- Pipelining breaks up a complex single-cycle operation into a sequence of shorter operations (pipelined stages), which are separated by memory elements and allow for higher clock rate. The drawback is the introduction of latency, which adds overhead for storage elements and can cause problems due to data dependencies in iterative algorithms.

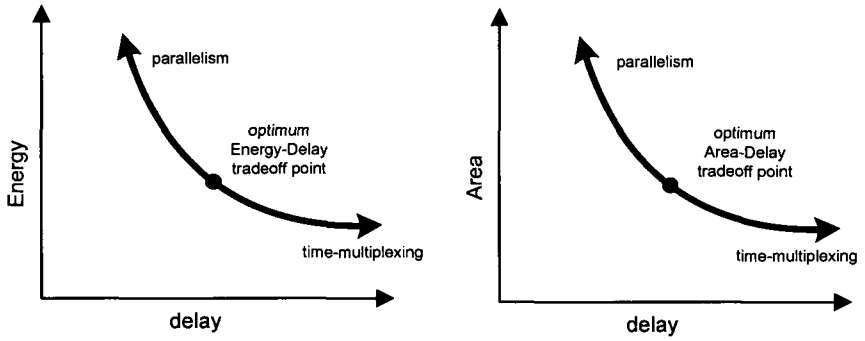


Figure 1.6 Energy-delay and area-delay trade off in digital circuits

1.4.4 Assessing VLSI circuit complexity

Once number and type of operations, numerical requirements, and necessary degree of parallelism are known; a rough estimate of the circuit complexity can be obtained from area and delay of the individual building blocks. In order to allow for a fair comparison between implementations across different silicon technologies, area is presented in equivalent gate number (GE) instead of square millimetres. One GE corresponds to the area of a two-input NAND gate in the respective technology. Accurate prediction of the delay of a circuit for different technologies is much more difficult especially with modern deep-submicron processes. Memory storage elements, often account for more than 50% of the area. Register-based memories used in small local memories and pipeline stages, consume 4-6 GE per bit. Area-optimized on-chip random access

memories (RAMs) concentrate the storage in large arrays. The area requirement of RAMs is about 0.5 GE/bit. RAMs only allow access to a few words at a time, while registers can be integrated directly for parallel access. RAMs also have a constant area overhead, which reduces the area efficiency of small RAMs.

1.4.5 Design flow

A Top-down design flow for mapping an algorithm into hardware can be summarized in the following steps:

- System studies and simulations.
- Algorithm selection, modification and development. In this stage a review of general category of algorithm will be done. From the most promising algorithm categories a review of the existing algorithms for each category will be performed and either new algorithm will be proposed for hardware implementation or modifications will be implemented in the current algorithms.
- Hardware architectural design:
    - Top level architecture exploration
    - Block level architecture
    - Data path design
    - Memory selection
    - Top-level integration
- Hardware design
    - Block level HDL design
    - Block level design synthesis and timing closure
    - Optimization for area, speed and power
    - Physical design
- Verification
- Hardware implementation and test

1.5 Thesis contributions

The main contribution of this dissertation is introduction of a low complexity dynamically configurable processing element (*MECU*) that can be used as the main processing unit for implementation of various MIMO detectors based on tree search

algorithms. *Low Complexity Cartesian Enumeration* (*LCCE*) algorithm is introduced for M-QAM constellation space. Three main tasks in tree search based detection algorithms are included in *LCCE* algorithm; i.e. child nodes enumeration, metric computations and sphere radius test. *LCCE* algorithm is designed as a configurable parallel algorithm specifically targeted for hardware implementation with low hardware complexity. The proposed *MECU* processing element is designed based on *LCCE* algorithm and can be dynamically reconfigured to support BPSK, QPSK, 16-QAM and 64-QAM modulated signals and is characterized by a low silicon complexity equivalent to 18K gates.

Figure 1.7 shows the logical dependency diagram between the architectures and algorithms developed in this dissertation along with the references to the corresponding section numbers in the thesis.

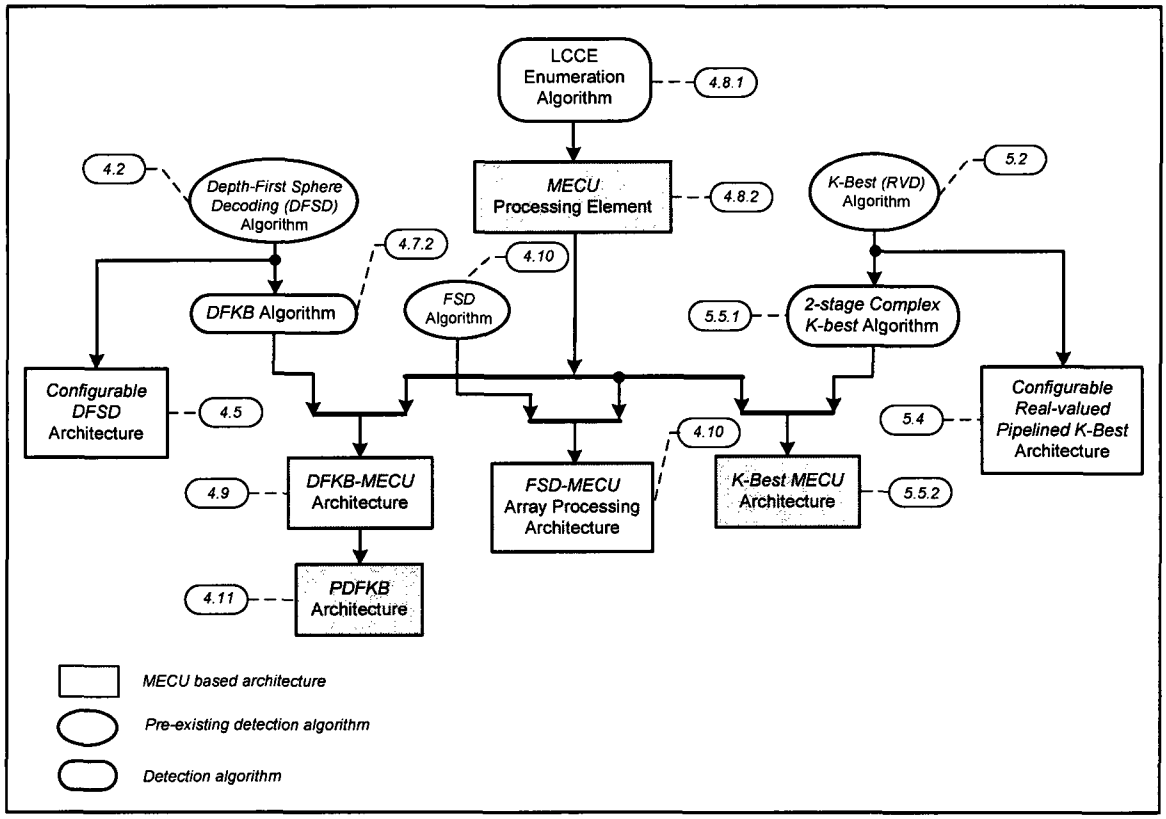


Figure 1.7 Thesis contributions-Logical dependency diagram

As it is shown in this figure, *MECU* processing element can be used to implement three independent groups of detector architectures. Each group is based on one of the three main categories of tree search algorithms; i.e. depth-first sphere decoding, fixed sphere decoding and K-best decoding algorithms. For each category of detection algorithm, it is

shown that *MECU* processing units can be connected in a specific architectural format in order to generate high performance detectors. As demonstrated in Figure 1.8, *MECU* processing elements can be utilized to form three different architectures: folded (recursive), array processing and pipelined to implement three groups of tree-search algorithms. This characteristic paves the way for the next generation of *multi-detection-algorithm* configurable detectors that can switch between different algorithms by rearranging the connections between the processing elements in order to optimize the algorithms to suite the operational requirements.

With respect to the depth-first sphere decoding, a multi-mode configurable implementation of this algorithm is used as the benchmark design for comparison with *MECU* based architectures. The *Depth-first-K-Best* (*DFKB*) sphere decoding algorithm is proposed as the improved version of the classical algorithm (Figure 1.7). The new DFKB algorithm overcomes the major shortcomings of the classical algorithm in throughput performance and memory requirements specifically in cases where the transmitter uses 64-QAM modulation in one of the channels.

Performance simulation results show that DFKB algorithm with level-based radius definition improves data throughput of the detector by 84% and reduces the number of required memory cells by 75% in 4x4 64-QAM modulation mode. A folded architectural implementation of this algorithm based on *MECU* processing element blocks (Figure 1.8(a)) provides a versatile detector choice for MIMO receivers with the ability to support a combination of antenna configurations (4x4, 3x3, 2x2) over a range of modulation modes (BPSK, QPSK, 16-QAM and 64-QAM) and achieves a maximum throughput of 672 Mbps.

Parallelism in recursive algorithms has always been a challenge which limits the use of the parallel connected processing units based on recursive algorithms. A solution for connecting two folded architectures for improving detector throughput is proposed that can result in smoother throughput variations with respect to signal to noise ratio in depth-first sphere decoding architectures.

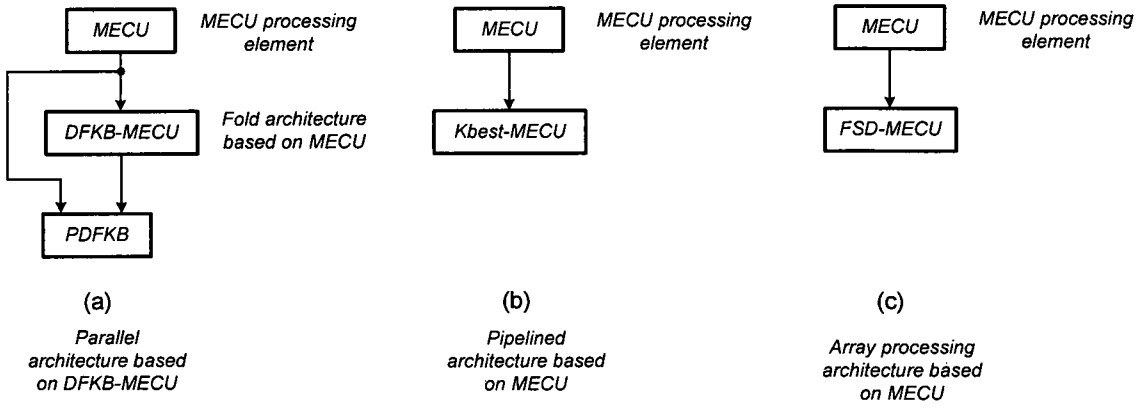


Figure 1.8 MECU based detector architectures

Parallel *PDFKB* architecture (Figure 1.8(a)) uses two folded DFKB detector units operating in parallel and shares a common data buffer with *odd-even* node distribution algorithm between the two detectors.

With respect to the implementation of K-best decoding algorithm, a configurable real-valued pipelined K-best detector architecture has been presented and implemented as the benchmark design (Figure 1.7). In this reference design, modulation mode (QPSK, 16-QAM and 64-QAM) and parameter K can be reconfigured in order to optimize detector operation over different operating conditions. It is shown that by applying a 2-stage node selection strategy for node elimination/selection and using the *LCCE* enumeration algorithm, the overall algorithmic and architectural complexity of complex-valued K-best detector can be reduced significantly. A complex-valued pipelined VLSI architecture for K-best algorithm using *MECU* processing elements is proposed (Figure 1.8(b)). The proposed architecture can support a range of modulation modes. The implementation results for a 4x4 (QPSK, 16-QAM and 64-QAM) detector compared to real-valued reference detector shows 53% decrease in silicon complexity and 75% increase in maximum clock frequency.

Fixed sphere decoding algorithm is a sub-optimal fixed complexity decoding algorithm that can be implemented in hardware using an array of processing elements. *MECU* processing element is used as the main processing node for the array processor and a VLSI architecture for a 4x4 (8-1-1-1) decoder is implemented (Figure 1.8(c)).

The implementation results in all three groups of algorithms show that *MECU* based architectures outperform the benchmark designs in silicon complexity as well as data throughput.

For validation purposes, a top-down verification methodology is followed. For this purpose, a multi-functional co-simulation-emulation environment is designed that can be used for system level simulations, RTL verification, hardware emulations and performance measurements.

In order to evaluate performance of DFKB-MECU architecture, a configurable 4x4/3x3/2x2 64-QAM/16-QAM/QPSK/BPSK ASIC detector has been designed and routed in CMOS 0.18 μm technology.

1.6 Organization of the thesis

Chapter 1 starts with an introduction to MIMO communications and presents the main motivations behind this dissertation in section 1.2. The main implementation challenges of MIMO detectors are discussed in section 1.3. Considerations regarding mapping algorithms to VLSI architectures are discussed in section 1.4. Thesis contributions and organization are listed in sections 1.5 and 1.6.

The baseband signal processing architecture for MIMO detection is introduced in Chapter 2. In section 2.2, the terminology used in reconfigurable computing is listed and the major definitions are reviewed. Configurability is the new design paradigm in future hardware systems; this issue is addressed in section 2.3. The baseband signal processing architecture for the future radio system is demonstrated in 2.4; followed by the general architecture for hardware accelerators presented in section 2.5.

In Chapter 3 a brief background review on MIMO channels and MIMO detection algorithms is presented. In 3.1 the mathematical channel model for MIMO systems is introduced followed by the system model for MIMO-OFDM communication systems in section 3.2. It is followed by a literature review of the state-of-the-art MIMO detection algorithms (3.3 and 3.4) and VLSI architectures for hard-decision and soft-decision MIMO detectors in section 3.6.

Chapter 4 presents a comprehensive study of sphere decoding algorithm. The classical sphere decoding algorithm is introduced in 4.2 followed by a brief review of enumeration

methods in section 4.3. Due to the recursive nature of depth-first sphere decoding algorithms, folded VLSI architectures are the natural choice for hardware implementation of this group of algorithms. In section 4.4 two different types of folded architectures for implementation of the sphere decoding algorithm are presented. A dynamically configurable architecture based on stack algorithm and classical sphere decoding for a 4x4/3x3/2x2 BPSK/QPSK/16-QAM MIMO system is presented in section 4.5. The limitations of sphere decoding algorithm for expansion to 64-QAM modulation and beyond are discussed in section 4.6. Algorithm modifications for improving detection and enumeration methods are proposed in section 4.7 which are the main motivation behind *LCCE* enumeration algorithm and the associated *MECU* hardware architecture proposed in section 4.8.

In the remaining sections of this chapter and Chapter 5, *MECU* processing element is used as the main processing unit for a number of MIMO detector architectures. *DFKB-MECU* architecture is presented in section 4.9. An array processing architecture for fixed sphere decoding algorithm is demonstrated in section 4.10. *PDFKB* architecture in section 4.11 is formed by parallel connection of two *DFKB-MECU* detection engines and further improves data throughput at low signal to noise ratios. In section 4.12 the implementation and simulation results for the proposed architectures are compared and the conclusions are drawn in section 4.13.

In Chapter 5, the implementation of K-best detection algorithm is reviewed and discussed. In section 5.2 the breadth-first K-best algorithm is presented followed by a discussion on the effect of channel variations on the performance of K-best detectors in section 5.3. Pipelined detectors based on real-valued decomposition are considered as the classical implementation of K-best decoders. A pipelined configurable architecture for a 4x4 QPSK/16-QAM/64-QAM MIMO system is demonstrated in section 5.4. The pipelined architecture is modified into a run-time configurable detector by applying a low complexity sorting network and a new processing element architecture that can be reconfigured for different modulation types. An asymmetrical tree consisting of a combination of modulation modes on transmitting antennas can be easily mapped to such architecture by separately reconfiguring each of the processing elements in the pipelined architecture.

The implementation of complex-valued *K*-best detectors for 16-QAM and 64-QAM modulation schemes has been a challenge. The main issue is the significant increase in the complexity of the sorting network dealing with complex-valued M-QAM constellation space. In section 5.5 complex-valued K-best detectors are considered by first introducing a low complexity 2-stage node selection algorithm in section 5.5.1. *MECU* processing units are used to design the processing elements in a 2-stage complex valued *K*-best architecture for a 4x4 QPSK/16-QAM/64-QAM MIMO system in section 5.5.2. Simulation and implementation results are presented and discussed in section 5.6.

In Chapter 6, verification and emulation platforms for different detector designs in this thesis are introduced. A top-down simulation environment based on *Simulink-Modelsim* co-simulation environment is presented in section 6.2. In general this test bench can be used for system level analysis, performance measurements as well as RTL verification. In section 6.3 this simulation environment is expanded to create an emulation platform for hardware emulations. The FPGA prototyping results are also presented in this section. The ASIC implementation results for DFKB-MECU architecture are demonstrated in section 6.4.

Chapter 7 presents conclusions and suggests some options for future research work.

Chapter 2 Baseband Processing

2.1 Introduction

The next generation wireless standards support for adaptive modulation and coding, user controlled quality of service, and cooperative networking requires the move to configurable computing machines for implementation of different processing stages in the radio system. An ideal computing system has the ability to tune the hardware systems to the needs of the operating conditions. In this chapter the baseband architecture for MIMO receivers will be introduced. In section 2.2 a review of reconfigurable computing is presented. Section 2.3 introduces configurability as a new design paradigm for baseband processors. In section 2.4, a baseband processing architecture for MIMO systems is presented which is based on DSP processors and hardware accelerators. In this architecture hardware accelerators perform the high processing power tasks such as MIMO detection. The main design guidelines for design of hardware accelerators are discussed in section 2.5.

2.2 Reconfigurable computing

The implementation platforms for reconfigurable computing machines can be categorized in four general groups; i.e. programmable processors, fine-grained arrays (e.g. FPGA), coarse-grained arrays, and dedicated ASIC processors [68]. The general purpose programmable processors are the most flexible hardware architectures and can be programmed to implement any mathematical or control algorithm. The energy efficiency of the programmable processors is generally low due to the large overhead of control layer associated with the operation of these devices.

Fine grained and coarse grained programmable arrays are reconfigurable hardware array structures that can be used for implementation of processing engines by programming and creating new connections between processing elements, memory blocks, logical function blocks and registers available in a programmable array device [75].

Application Specific Integrated Circuits (ASICs) based processors are dedicated VLSI circuits optimized for a particular set of tasks forming an algorithm with limited adjustments. ASICs have the advantages of low power dissipation and high clock

frequency but limited flexibility. In the following sections the terminology used in reconfigurable computing will be reviewed.

2.2.1 Adaptive system

An adaptive system is capable of maintaining and/or improving its performance under internal or external disturbances or variations. Such disturbances or variations can occur during fabrication, under faults, physical parameter degradations, environment variations, external interferences, users' application preferences, requirements modifications, and performance-resources trade-offs [74].

2.2.2 Flexible hardware

Flexibility is defined as the ability of a hardware system to adapt its parameters to various operating conditions and modes. Operating conditions are internal and external effects that are affecting the performance of hardware system such as noise power in a communication system. Operating modes are defined as eligible designed modes of operation.

A flexible design not only should execute specific algorithms, but also should be able to readjust itself under varying operating conditions. As an example we can mention power consumption of mobile processing devices. In such devices power consumption should be optimized based on different factors such as source of power, battery charge levels and throughput requirements. A mobile processing system should continuously readjust itself to find the optimum power consumption point.

As mentioned before, flexibility can be achieved by using both hardware and software techniques. Dedicated ASIC hardware processing systems are designed for high performance low power processing applications, providing a lower energy per processed bit compared to programmable processors and FPGA based processing systems. The main challenge in designing flexible processing architectures is the trade-offs between processing power and flexibility required by the processing application.

2.2.3 Dynamically configurable system

A dynamically configurable system is characterized by the ability to rapidly alter the functionalities of its components and interconnections between them to serve the new operating condition. Based on the above definition, a dynamically configurable hardware is an adaptable and flexible hardware with the ability of run-time reconfigurations. The

key to this ability is speed of reconfiguration. In static reconfiguration, the configuration rate ranges from seconds to hours. The reconfiguration rate in dynamic reconfiguration on the other hand varies from microseconds to seconds.

2.2.4 Homogenous and heterogeneous architectures

Homogenous architectures are implemented based on only one type of processors such as general purpose processors or digital signal processors (DSP). Due to computationally intensive tasks in baseband processing a homogeneous architecture solely based on programmable devices cannot provide the required computational power required by the high demand tasks. Moreover the energy efficiency of programmable devices is low for battery-powered wireless devices.

Heterogeneous hardware platforms on the other hand consist of different types of processing elements and also with different granularities [33]. In a heterogeneous platform, different processors are interconnected by a network-on-chip (NoC). In general both processing elements and NoC can be dynamically configurable which means that the communication channel linking the processing elements can also be changed at run-time.

2.2.5 Reconfigurable hardware

The idea of configurable computing machines using reconfigurable hardware has been around since 1960s [43]. The first Field Programmable Gate Arrays (FPGAs) became available in 1985 and since then they have been used as a platform for high performance computing. The ultimate vision was for a technology that enables implementation of dedicated hardware processors without the cost, delays and risk associated with full custom design.

The granularity of a reconfigurable hardware is determined by the width of the data path in the components building the architecture. In *fine-grained* architectures, functionality of the architecture is specified at the bit-level and programmable interconnects are used as wires to connect components. Fine-grained architectures are efficient for complex bit-level computations. FPGA devices such as Altera Cyclone [56] and Xilinx Virtex [44] are examples of fine-grained reconfigurable architectures.

Coarse-grained reconfigurable hardware on the other hand contains word-level functional units such as multipliers. Programmable interconnects in this case are buses connecting word-level functional units [67]. Hybrid architectures containing both fine-

grained and coarse-grained elements are suitable for efficient implementation of both bit- and word-level computations. Altera Startix II [25] is an example of hybrid reconfigurable architectures.

2.2.6 Hardware accelerator (HA)

Hardware acceleration is defined as the use of dedicated hardware blocks to perform specific processing functions in a signal processor. Dedicated hardware blocks that performs acceleration, when located in a separate module from the microprocessor, are commonly referred to as *hardware accelerators*. The main reason for including hardware accelerators is the added parallelism and higher performance provided by the hardware compared to the sequential software run on a programmable processor. Due to inherent concurrency within a hardware processing system, dedicated hardware processors are much faster than programmable processors hence hardware accelerators are designed for computationally intensive processing tasks such as motion estimation in MPEG2 [33].

2.3 Configurability a new design paradigm

One of the most challenging aspects in designing computing systems is the balance between flexibility versus efficiency of the system. Future wireless communication devices are multimode and multifunctional systems that can operate under different standards. Configurability becomes a major objective in design and operation of future wireless systems. The computing system has to adapt (*run-time* adaption) and readjust itself in response to changes in environmental conditions (varying noise figure, cell traffic ...) as well as to changing operating modes variations (bandwidth, traffic data pattern, and quality of service (QoS)).

One of the main reasons for applying configurable hardware in wireless terminals is to support multiple communication standards in order to increase mobility and reduce operating cost of the device. *Standard level* configurability provides the ability to switch between wireless communication standards and has a major impact on the complexity of digital signal processing in wireless terminals. The second-level of configurability is *algorithmic level* which is the ability of selection signal processing algorithms to implement certain functionality in baseband processing. For every baseband processing

task there are a number of algorithmic choices that have their own optimized range of functionality [69].

The functionality of DSP tasks are often specified by the wireless standards and the specific algorithmic details are usually left for implementation. Baseband processors can either select an algorithm from a set of predefined algorithms that best suit the operating and environmental conditions or change parameters of algorithms in order to optimize performance. In either case, dynamic reconfiguration of hardware is required in order to achieve both algorithmic and standard level configurability even though the algorithm-parameter reconfiguration is being addressed more frequently and requires higher reconfiguration rates compared to standard level reconfiguration.

2.4 Baseband processor architecture

Baseband signal processing in radio transmitter/receiver systems is referred to as the signal and data processing activities that are located between A/D and D/A converters on one side and digital data samples on the other side. From circuit design perspective; baseband processing is the digital part of physical layer design in a transceiver system.

In software defined radio (SDR) platforms, an embedded programmable processor (general microprocessor or DSP) controls the main functionalities in baseband processing and provides an interface to MAC layer. In typical MIMO transceiver architectures; the following tasks and functional blocks can be identified as part of the baseband processing.

- Channel estimation
- Channel pre-processing
- Frontend processing
- Pilot insertion
- FFT / IFFT
- Guard insertion (GI) / Guard removal (GR)
- Synchronization
- Phase tracker
- Symbol mapper
- MIMO encoder

- MIMO detector
- Channel encoder and decoder
- Interleaver / De-Interleaver
- Filtering, sample rate adjustment and carrier frequency adjustment

Symbol mapper, MIMO encoder, and MIMO detector (decoder) are all part of the functionalities which are related to modulation and demodulation/detection of symbols. The following design challenges can be identified with regards to baseband processors:

- Increasing design complexity due to more complex signal processing algorithms which in turn increases the overall design cost and product's time-to-market.
- Flexible designs and the cost associated with flexible ASIC design.
- Power consumption in battery powered devices and power efficiency of programmable processors.

Heterogeneous baseband processor architecture based on RISC (Reduced Instruction Set Computer) processor core and a number of configurable hardware accelerators as shown in Figure 2.1 can provide the level of computational power demanded by complex signal processing algorithms while maintaining the high level of flexibility required in baseband processors. Such approach to the implementation of baseband processors improves the overall performance of the processor significantly by increasing parallelism in the system.

Hardware accelerators are small configurable processing engines that can be optimized for performance and power consumption while maintaining certain level of flexibility. The programmable processor can focus on data control operations, reconfiguring individual hardware accelerator blocks and executing smaller algorithms suitable for software implementations. The implementation platform for such architecture can be either ASICs or FPGA based on the application needs.

Hardware accelerators, memories and external interfaces are connected to the processor core via an interconnect network. This network can be either a crossbar switch or a standard bus interface used for on chip networking [76]. Direct one-to-one connections between hardware accelerators can be utilized in order to reduce interconnect network traffic and access time. An example of such direct interconnect can be seen between channel preprocessing and detector block in Figure 2.1.

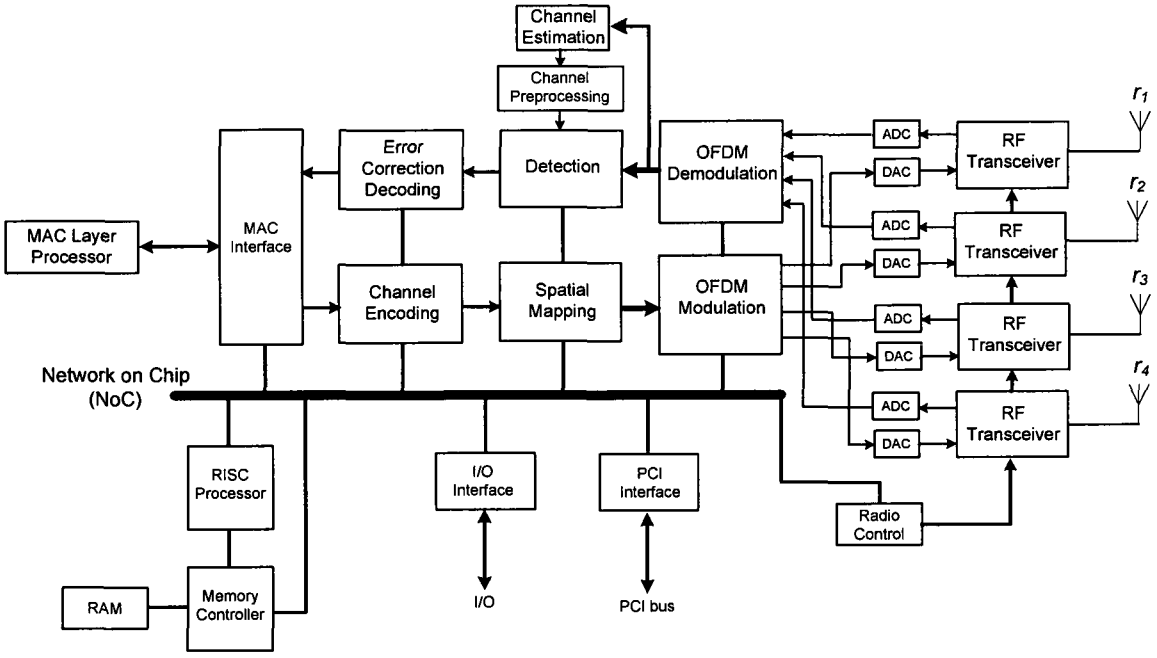


Figure 2.1 Baseband processor architecture

2.5 Hardware accelerator design guidelines

In Figure 2.2 general architecture of hardware accelerators (HA) has been depicted. The role of *Input/Output Interface* blocks is to provide the communication link between HA and the outside world. Input data vectors can be either configuration parameters or data packets. Configuration parameters are input information that are needed for reconfiguration of individual blocks as well as fixed data inputs that are needed for data processing such as communication channel parameters. The output of each processing element (PE) can be stored in a *Buffer* or connected to the *Switch Fabric*. Switch fabric is a configurable cross bar switch which connects inputs and outputs of the processing elements. The internal buffer structures can be either register based or RAM based. Register based buffers have faster access time compared to RAMs but generally have higher area and consume more power. *Control Unit* configures switch fabric connections. The main design factors to be considered while designing hardware accelerators can be summarized as follows:

1. Area and power cost of HA design compared to overall area and power budget of baseband processor.

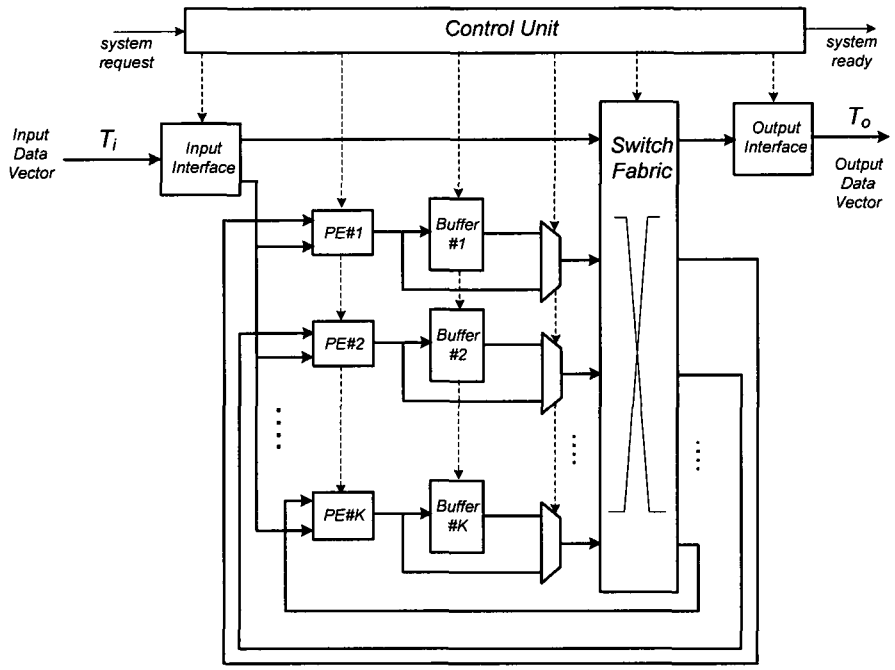


Figure 2.2 Hardware accelerator architecture

2. Computational performance of HA compared to the number of run cycles in a pure software based implementation.
3. Dynamic configurability of applications which require real-time reconfigurations and multi-mode operations.
4. Scalability of design, i.e. the ability of hardware accelerator block to be connected to similar blocks in order to expand system dimensions.
5. Reusability of hardware accelerators for different standards.

Additional power saving measures such as clock gating can be applied to further reduce power consumption in configurable hardware accelerator. Hardware accelerator architectures based on homogenous processing element blocks can be partitioned into sub-blocks for power gating.

Chapter 3 Overview of MIMO Detection Algorithms

3.1 MIMO channel model

In a MIMO communication system with N_t transmitter and N_r receiver antennas ($N_t \geq N_r$), each of the N_r receiver channels receives signal components from each of the N_t transmitters, including both line-of-sight and reflected components. Figure 3.1 shows a model of multi-path transmission system. The system model for a MIMO system in a flat-fading channel can be described by the following equation:

$$\mathbf{y} = \mathbf{Hs} + \mathbf{n} \tag{3-1}$$

Where $\mathbf{y}$ is $N_r x 1$ vector of received symbols, $\mathbf{H} = [h_{ij}]$ is $N_r \times N_t$ equivalent flat-fading channel matrix, $\mathbf{s} = [s_1, \ldots, s_{Nt}]^T$ is $N_t x 1$ vector of transmitted symbols. The $N_r x 1$ vector $\mathbf{n}$ represents complex thermal noise as independent identical distributed (i.i.d) additive complex Gaussian noise at the receiver with zero mean and variance σ^2.

For a narrow band transmission, e.g. single channels in an OFDM system, the receiver has to solve the multi-path transmission model described by equation (3-1) in order to recover transmitted symbols.

Transmit vector $\mathbf{s}$ corresponds to a binary vector $\mathbf{b}$, containing N_t Q_b bits, where Q_b is the number of bits per symbol in the complex constellation Λ containing $P_c = 2^{Qb}$ constellations.

The task of MIMO detector is to recover $\mathbf{s}$ from $\mathbf{y}$ by solving equation (3-1). The *uncoded transmission rate* is $R = N_t Q_b$ bits per channel use (bpcu). Wideband communication systems can be reduced to a set of narrowband MIMO system by using OFDM modulation and the above narrowband model can also be applied to receivers in wideband systems [46].

3.2 MIMO-OFDM systems

Orthogonal Frequency Division Multiplexing (OFDM) is a modulation scheme that is robust against interference arising from multipath propagations.

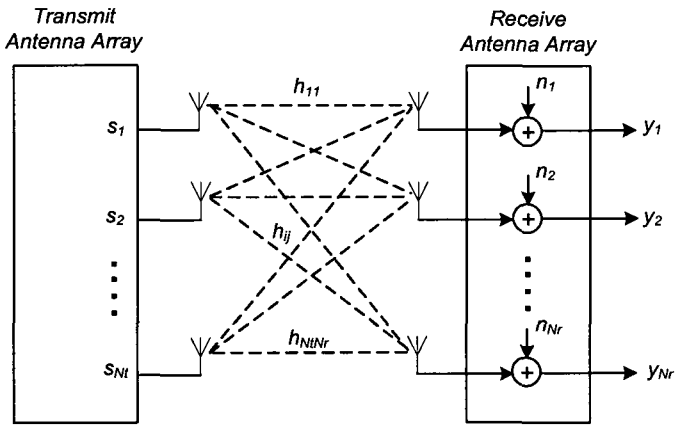


Figure 3.1 Multi-path transmission model

Wireless local area networks (WLAN) such as IEEE 802.11a and IEEE 802.11g are examples of OFDM based systems. Many upcoming new standards for high throughput wireless communication such as IEEE 802.11n and IEEE 802.16 rely on a combination of MIMO with OFDM called MIMO-OFDM technique. In such systems, received signals can be seen as a linear combination of the signals from all the transmitting antennas. Figure 3.2 shows structure of a MIMO-OFDM transceiver. Transceiver chain consists of MIMO blocks including modules for estimating channel matrix as well as MIMO detection block. There are also OFDM related blocks such as IFFT/FFT and guard interval insertion/removal blocks. In a packet based MIMO-OFDM system with N_t transmitter and N_r receiver antennas, the wide band channel is split into orthogonal subchannels (tones) so that modulation and MIMO detection can be performed on a tone-by-tone basis. After OFDM modulation at the transmitter and demodulation at the receiver (with the assumption of proper synchronization, frequency offset estimation and adding cyclic prefix), the received vector $\mathbf{y}[k,t]$ is given by the following equation

$$\mathbf{y}[k,t] = \mathbf{H}[k]\mathbf{s}[k,t] + \mathbf{n}[k,t] \tag{3-2}$$

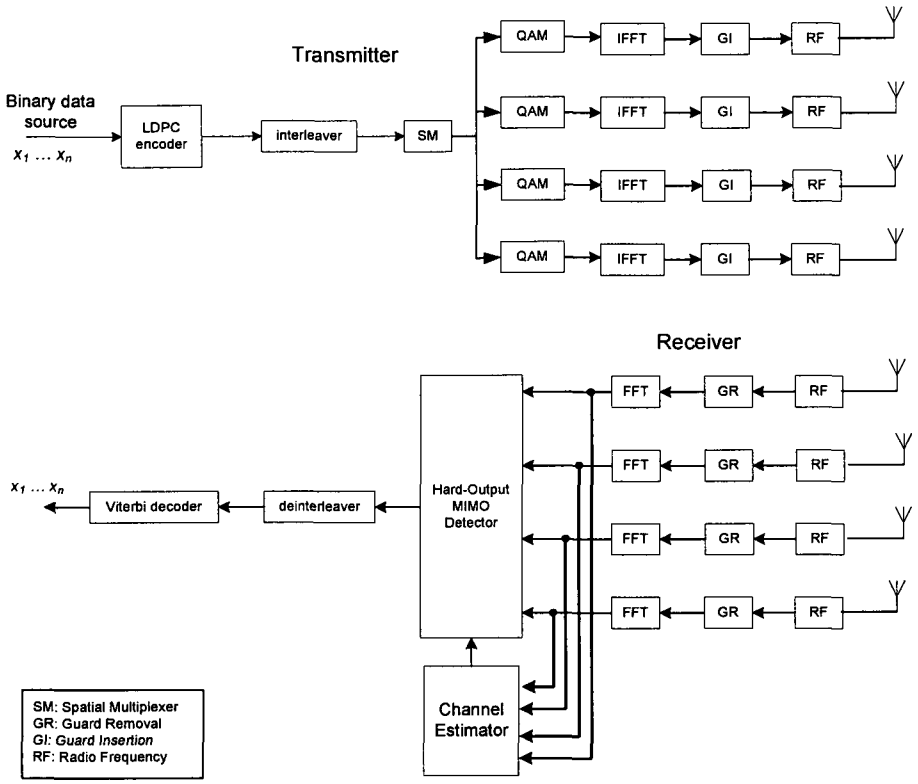


Figure 3.2 MIMO-OFDM transceiver chain

Where k is the number of the data carrying tones in OFDM modulation. As an example in a 256-point FFT system; k is less than or equal to 256, $\mathbf{s}[k,t]$ is the N_t-dimensional signal vector transmitted at time t on the kth tone of the OFDM signal, $\mathbf{H}[k]$ is a N_txN_r dimensional matrix that models MIMO channel for the kth tone and $\mathbf{n}[k,t]$ models the thermal noise in the system as an i.i.d. complex Gaussian with variance σ^2 per complex dimension. Different tones have different channel models that should be estimated at the beginning of transmission of each data frame.

In this thesis, the focus is on one channel use for each tone and as a result equation (3-1) will be used instead of equation (3-2) which represents the general MIMO-OFDM model. During the initial training phase, receiver performs channel estimation using pilot tones and obtains required information to form $\mathbf{H}[k]$. In preprocessing stage $\mathbf{H}[k]$ is processed to calculate all the required matrices for detection. Throughout this dissertation it is assumed that the preprocessed channel parameters are available to MIMO detector.

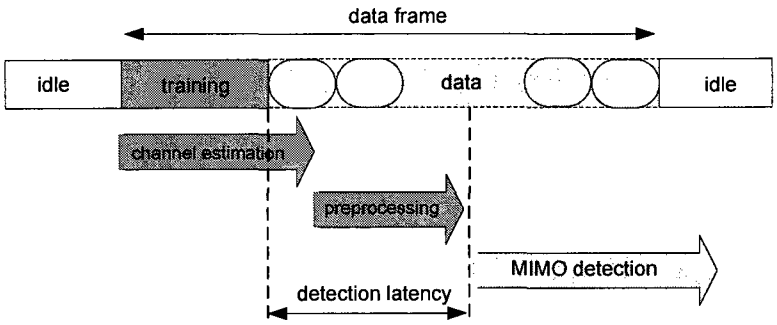


Figure 3.3 MIMO detection process in packet based MIMO-OFDM systems [28]

The delay incurred by preprocessing is translated directly into detection latency as shown in Figure 3.3.

3.3 MIMO Detection

MIMO detectors fall into two main categories: hard-decision (hard-output) detectors and soft-decision (soft-output) detectors. Hard-output detectors provide hard decisions of the transmitted bits. A channel decoder such as Viterbi decoder can be used in conjunction with a hard-decision detector to perform error correction. The block diagram in Figure 3.2 shows a hard-decision MIMO detector. On the other hand soft-output detectors provide a-posteriori probability (APP) information about each bit, which can be used to realize either iterative detection in conjunction with an outer channel decoder such as low-density parity check (LDPC) decoder [7][48]. Figure 3.4 shows the block diagram of an iterative decoder.

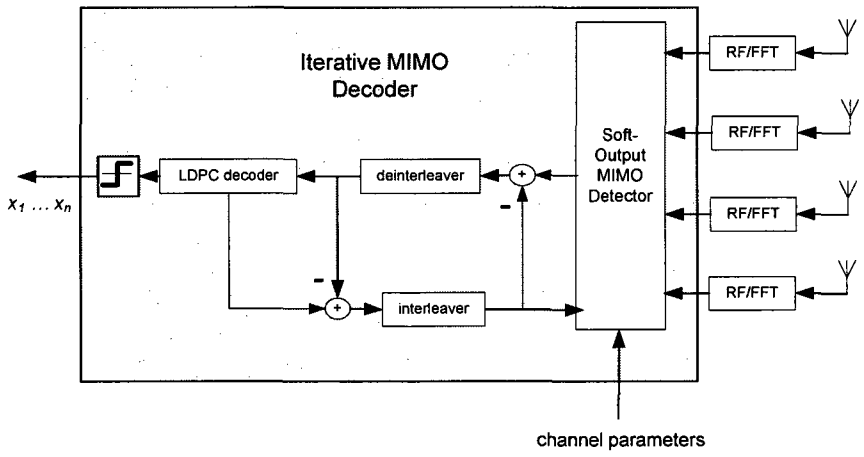


Figure 3.4 Soft-output MIMO decoding

3.4 MIMO detection Algorithms

The role of MIMO detectors is to recover transmitted symbols by solving MIMO system model in equation (3-1). In this section we briefly review and compare some of the algorithms that exist for solving this equation. These algorithms can be used in both hard-output and soft-output decoders.

Figure 3.5 shows the main divisions of algorithms for MIMO detection. Among the available algorithms for MIMO detection, one can identify two main categories of algorithms: Maximum likelihood (ML) methods and non-ML methods. ML based algorithms have low BER and high complexity and are often divided into two groups of algorithms:

- Exhaustive search.
- Sphere decoding.

Non-ML methods can be divided into three main groups of algorithms:

- Linear detection method.
- Successive interference cancellation (SIC) techniques.
- Sub-optimal tree search methods.

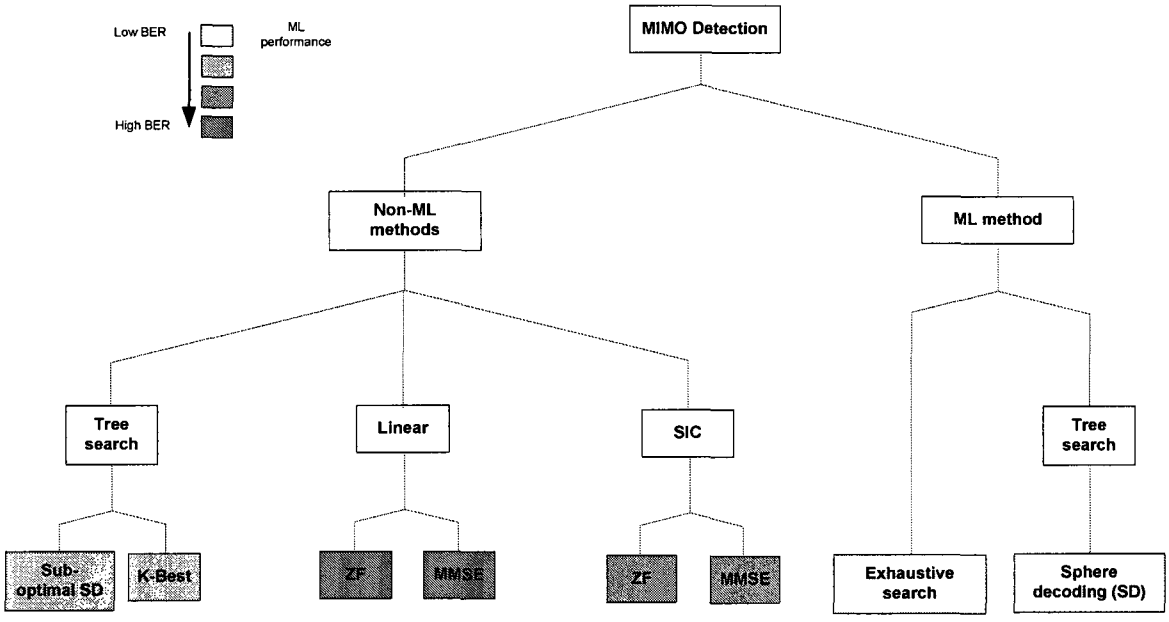


Figure 3.5 MIMO detection algorithms

MIMO detection algorithms can also be categorized based on their BER performance. In this case BER curve deviation from ML BER curve can be used as a measure of BER

performance compared to the optimum detection performance. In Table 3.1, the BER deviation range for each performance group and examples of algorithms in each group has been listed.

Table 3.1 BER performance measure for MIMO detection algorithms

BER Performance	Deviation from ML curve @ 30 dB SNR level	Algorithm Examples
ML	0 dB	Exhaustive search Sphere decoding
Sub-optimal (quasi-ML)	0 ~ 2 dB	Sphere decoding-early termination Sphere decoding- l^{∞} norm K-best
High Performance	2 ~ 6 dB	K-best
Medium Performance	6 ~ 10 dB	V-Blast
Low Performance	> 10 dB	Linear ZF Linear MMSE

3.4.1 Tree Search Strategies

A tree graph can be used to represent hierarchical data structure in a graphical form. It is named *tree* (or *search tree*) because the graph looks like a suspended tree, i.e. the *root* is at the top and the *leaves* are at the bottom. Any *node* in the tree contains a value or a condition and has zero or more *child nodes*, which are located below it in the graph. A node that has a child is called *parent node*. A node has only one parent. The top node in a tree is called *root node*. All the nodes can be reached from root by following edges or links. Nodes at the bottom level of the tree are called *leaf nodes* (or leaves) and they do not have any children. The lines connecting nodes are called *branches*.

All possible combinations of transmitted symbols in a MIMO communication system can be mapped to a tree graph as illustrated in Figure 3.6 for a 3x3 BPSK MIMO system. The root of this tree is on level N_t+1 and each node at level i corresponds to a *partial candidate vector symbol* $\mathbf{P}_i$. The leaves on the first level show all the possible combinations of transmitted symbols.

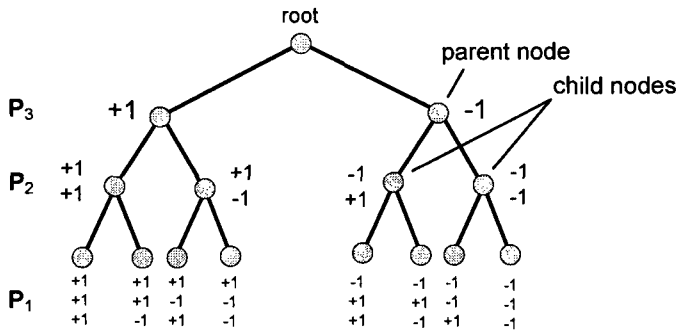


Figure 3.6 Search tree for a 3x3 BPSK MIMO system

Tree search methods are either depth-first or breadth-first. In this section we review these search strategies that are the basic idea behind tree search based algorithms for solving equation (3-1).

Depth-first Tree Search (DFTS)

Depth-first tree search starts at the root of the tree ($i= N_t+ 1$) and progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or it hits a node that has no children. Then the search backtracks, returning to the most recent node it had not finished exploring (Figure 3.7). It is also possible that a constraint violation causes backtrack to a higher level. The search continues until all the nodes are visited once.

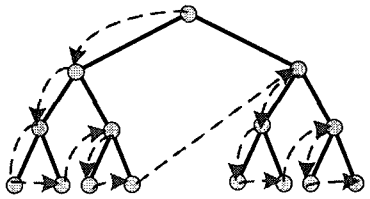


Figure 3.7 Depth-first tree search

Breadth-first Tree Search (BFTS)

Breadth-first tree search is a non-recursive search algorithm that begins at the root of the tree ($i= N_t + 1$) and traverse tree only in forward direction. A selected group of nodes on every level are called *admissible* nodes. On the ith level of the tree, it explores all the admissible nodes and considers their associated children to construct a new set of admissible nodes on the next level before it proceeds. The search continues until the answer is found or it reaches the first level of the tree (i=1).

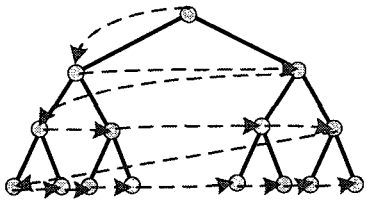


Figure 3.8 Breadth-first tree search

3.4.2 Real-valued Decomposition (RVD)

The constellation symbols in M-QAM modulation are represented by complex values. It is possible to decompose equation (3-1), to real and imaginary parts and only deal with real equations. The real-valued decomposition can reduce complexity of metric calculations. In real-valued constellations, the number of child nodes for each parent node P_r is much smaller than P_c ($P_c = P_r^2$). The drawback of real-valued decomposition is that the depth of a real-valued tree will be doubled and as a result the average processing time for each symbol in DFTS based algorithms increases. It is possible to transform the N_t-dimensional complex equation (3-1) to an equivalent $2N_t$-dimensional real-valued representation as follows [6].

$$\begin{bmatrix} \Re\{\mathbf{y}\} \\ \Im\{\mathbf{y}\} \end{bmatrix} = \begin{bmatrix} \Re\{\mathbf{H}\} & -\Im\{\mathbf{H}\} \\ \Im\{\mathbf{H}\} & \Re\{\mathbf{H}\} \end{bmatrix} \cdot \begin{bmatrix} \Re\{\mathbf{s}\} \\ \Im\{\mathbf{s}\} \end{bmatrix} + \begin{bmatrix} \Re\{\mathbf{n}\} \\ \Im\{\mathbf{n}\} \end{bmatrix} \tag{3-3}$$

Decomposition essentially transforms the original search tree into a tree that is twice as deep, but has fewer children per parent nodes. The entries of equation (3-3) are defined by real-valued amplitude modulated constellation points, which constitute a finite lattice. RVD deals with real value constellation points and has less computational complexity than the complex valued tree search in breadth-first search algorithms.

3.4.3 Maximum likelihood detection

Maximum Likelihood (ML) detector solves equation (3-4) to find transmitted symbols.

$$s_{ML} = \arg\min_{s \in \Lambda} \|\mathbf{y} - \mathbf{Hs}\|^2 \tag{3-4}$$

Where Λ is the set of all possible transmitted symbols. The channel matrix is assumed to be perfectly known to the receiver. By triangularizing channel matrix **H** using *QR* decomposition algorithm we can further simplify equation (3-1) [5]:

$$s_{ML} = \arg\min_{s\in\Lambda} \left\| \mathbf{y}_q - \mathbf{Rs} \right\|^2 \tag{3-5}$$

$$\mathbf{H} = \mathbf{QR} \tag{3-6}$$

where **Q** is orthogonal and **R** is an upper triangular matrix

$$\mathbf{y}_q = \mathbf{Q}^{\mathbf{H}}\mathbf{y} \tag{3-7}$$

expanding vector norm in (3-5) yields

$$s_{ML} = \arg\min_{s\in\Lambda} \sum_{i=1}^{N_t} \left| y_{qi} - \sum_{j=i}^{N_t} R_{ij} s_j \right|^2 \tag{3-8}$$

starting at level $i=N_t$ (3-8) can be solved recursively as follows:

$$T_i(\mathbf{P}_i) = T_{i+1}(\mathbf{P}_{i+1}) + \left| e_i(\mathbf{P}_i) \right|^2 \tag{3-9}$$

with

$$e_i(\mathbf{P}_i) = y_{qi} - \sum_{j=i}^{N_t} R_{ij} s_j \tag{3-10}$$

$T_i(\mathbf{P}_i)$ is known as *node metric* (metric) or *Partial Euclidean Distance* (PED) and it is assumed that $T_{N_t+1}(\mathbf{P}_{N_t+1}) = 0$.

Equation (3-10) can be broken down into two equations as demonstrated in equation (3-11).

$$\begin{aligned} e_i(\mathbf{P}_i) &= C_{i+1} - R_{ii} s_i \\ C_{i+1} &= y_{qi} - \sum_{j=i+1}^{N_t} R_{ij} s_i \end{aligned} \tag{3-11}$$

This new formulation of $e_i(\mathbf{P}_i)$ based on C_{i+1} will be later used in section 4.8 for derivation of a new enumeration algorithm.

Equation (3-4)) can be solved using either an exhaustive search of all possible combinations of transmitted symbols (section 3.4.3.1) or by applying an iterative tree search methodology (sections 3.4.3.2 and 3.4.4.3). Figure 3.9 compares BER performance of linear, SIC and maximum likelihood detectors. Maximum likelihood detectors have the best BER performance but they are also the most complex ones. There are a number of algorithms that reduce the complexity of ML decoder and we will review them later in this chapter.

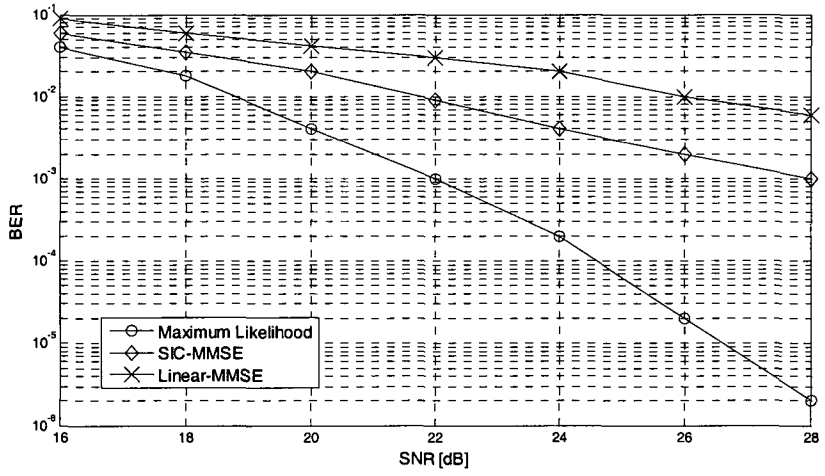


Figure 3.9 BER performance of linear, SIC and ML decoders

3.4.3.1 Exhaustive search detector

In exhaustive search detectors, s_{ML} is determined by exhaustively examining all the possible 2^R combinations of transmitted symbols where R is the uncoded transmission rate. The search for transmitted symbols can be performed using either depth-first or breadth-first tree search. Since in exhaustive search, detector evaluates all the possible combinations of transmitted symbols, the complexity of both tree searching methods is equal. The number of possible transmitted symbol for a 4x4 QPSK system is 2^8=256 and for a 4x4 16-QAM and 64-QAM systems this number raises to 2^{16}=65536 and 2^{24}=16777216 respectively. As the complexity of algorithm grows exponentially with respect to R, an implementation of this algorithm is only practically feasible in the low transmission rates (R≤8) [22].

3.4.3.2 Sphere detector (SD)

Sphere decoding algorithm has its origins in the work published by Pohst as a method for computation of lattice vectors with minimal length [2]. Its application as a detector for multiple antenna channels appears in [6] and a complex version of it appears in [7]. This algorithm can achieve optimal or near optimal ML performance with reduced complexity. The fundamental idea in sphere decoding is to reduce the number of candidate symbols that needed to be considered in the search for ML solution in equation (3-8). In this algorithm the search is limited only to those candidate vector symbols **s** for which **Hs** lies

inside a sphere with radius r around the received point **y**. We can express this condition by the following inequality:

$$\|\mathbf{y} - \mathbf{Hs}\|^2 < r^2 \tag{3-12}$$

Equation (3-12) is called *sphere constraint* (*SC*) or *sphere radius test*. This approach can significantly reduce the average search complexity, while the BER performance remains close to exhaustive search detector (ML performance). Throughput of this type of detector is variable and depends on the initial choice of sphere radius (r) and channel conditions. Furthermore if r is too small, the sphere constraint may not include the ML solution and if r is too big, detector throughput will be affected by the increased number of visited nodes.

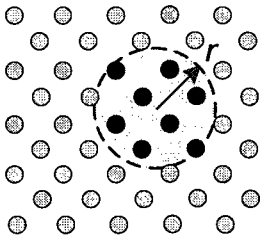


Figure 3.10 Sphere constraint

In section 4.2 the depth-first sphere decoding algorithm will be revisited with more details.

Average throughput

The average throughput (*data rate DR*) of sphere detection algorithm for a depth-first search in bits per second can be calculated using the following equation

$$DR = \frac{N_t Q_b f_{clk}}{D_{avg}} \tag{3-13}$$

In this equation D_{avg} is the average number of processing cycles for each of the received symbol vectors, Q_b is the number of bits per symbol, N_t is the number transmitting antennas and f_{clk} is the operating clock frequency. The average throughput can also be expressed by the average number of visited nodes in the search tree and the processing cycles per node. In equation

(3-13), f_{clk} is limited by the hardware physical constraints or critical timing path of the design while D_{avg} is a factor of efficiency of tree search algorithm. The highest throughput can be achieved by joint optimization of the two parameters.

Radius reduction

Sphere radius in sphere decoding algorithm can be reduced dynamically to limit number of visited nodes in depth-first tree search. Once a valid solution has been identified, we can update the value of sphere radius to the value of the PED associated with this solution and continue the tree search with the updated radius value. Without radius reduction, the order in which the children of a parent node are explored in a depth-first search has no influence on the overall complexity of the search and the only parameter that determines the complexity of the search is the initial sphere radius.

Schnorr-Euchner (SE) Ordering Rule

The basic idea behind Schnorr-Euchner (SE) ordering rule [4] is that by giving preference to the nodes with smaller metrics, one can expect to find leaves for which $\|\mathbf{y}-\mathbf{Hs}\|^2$ is smaller, earlier in the search which eventually leads to faster shrinkage of sphere radius and finding the transmitted symbols. The result of such ordering is that a smaller number of cycles is needed to find the solution of equation (3-8). Following SE ordering rule, the next parent node to be considered is the current child node that has the minimum metric value.

The drawback of SE ordering is that it requires an ordering of the children of parent nodes according to their associated metrics, which causes additional complexity specifically in higher order modulation schemes. The process of efficiently identifying and enumerating the admissible children of a parent node in SE ordering is one of the most critical parts for the efficient implementation of depth-first sphere and K-best decoding. Enumeration algorithms will be reviewed in section 4.3.

3.4.4 Non-ML (sub-optimal) Detection

3.4.4.1 Linear Detection

A straightforward approach to solve equation (3-1) and recover **s** from **y** is to consider MIMO system as an unconstrained linear estimation problem which can be solved by least squares zero forcing (ZF) or minimum mean squared error (MMSE) methods. The linear estimator can be used to estimate transmitted vector **s** from received vector **y**.

$$\hat{\mathbf{s}} = \mathbf{G}\mathbf{y} \tag{3-14}$$

Where **G** is the linear estimator matrix. Linear detectors have the lowest BER performance but are also associated with the lowest complexity order [47]. Computation of **G** is carried out only when the channel changes and is referred to as preprocessing.

Zero Forcing (ZF)

Zero Forcing (ZF) tries to null out channel interference by directly inverting the channel matrix [41]. The least-squares estimator G is defined as

$$\mathbf{G} = \left(\mathbf{H}^H\mathbf{H}\right)^{-1}\mathbf{H}^H \tag{3-15}$$

In the case of equal transmitter and receiver antennas (N_r = N_t) we have $\mathbf{G} = \mathbf{H}^{-1}$ and $\hat{\mathbf{s}} = \mathbf{s} + \mathbf{n}_{ZF}$ where $\mathbf{n}_{ZF} = \mathbf{G}\mathbf{n}$. In this case the effective channel matrix between transmitter and receiver $\tilde{\mathbf{H}}$ is the identity matrix **I**. The drawback of the ZF estimator is the increase in the additive noise as demonstrated by $\mathbf{n}_{ZF}$.

Minimum Mean Square Error (MMSE)

A drawback of zero forcing is that it nulls out the interference without considering the fact that the noise power could boost up significantly in the estimation. The MMSE estimator minimizes the overall expected error by considering the effect of noise [41]. The optimum solution for this estimator is

$$\mathbf{G} = \left(\mathbf{H}^H\mathbf{H} + N_t\sigma^2\mathbf{I}\right)^{-1}\mathbf{H}^H \tag{3-16}$$

The estimated vector can be obtained by replacing (3-16) in (3-14) and (3-1).

$$\hat{\mathbf{s}} = \tilde{\mathbf{H}}\mathbf{s} + \mathbf{n}_{MMSE} \tag{3-17}$$

$$\mathbf{n}_{MMSE} = \mathbf{G}\mathbf{n} \tag{3-18}$$

In equation (3-17), $\tilde{\mathbf{H}} = \mathbf{G}\mathbf{H}$ is the effective channel matrix after MMSE equalization.

3.4.4.2 Successive Interference Cancellation (SIC)

Successive Interference Cancellation (SIC) is based on the previously described linear estimation algorithms. In this method a nonlinear interference cancellation stage partially exploits the knowledge that entries of transmitted vector **s** have been chosen from a finite set of constellation points. The symbols of the parallel data streams are no longer detected at once, instead they are considered one after another and their contribution (after slicing and demodulation) is removed from the received vector before proceeding

to detect the next stream. Compared to linear detection SIC achieves an increase in diversity order after each iteration. In SIC the overall BER performance is dominated by the stream that is detected first and error propagation has considerable impact on the performance of the subsequent streams. Hence the detection order is important for achieving good BER performance. For unordered SIC the complexity order is almost the same as linear detection schemes [47].

Ordered SIC Detection (OSIC)

Ordering aims at reducing the likelihood of detection errors by attempting to identify the streams that are most likely to yield no detection errors in the first iterations. The V-BLAST ordering algorithm [45], finds the ordering which is optimum with respect to BER performance when only the nature of the channel is taken into account. The algorithm determines the order in such a way that the noise on estimated stream is minimized in every iteration through the algorithm and consequently the error probability of the subsequent slicing operation is also minimized. The complexity order of V-BLAST is more than linear detection due to the increased complexity in preprocessing.

3.4.4.3 K-best detector (KB)

K-best algorithm is a breadth-first search algorithm with sub-optimal solution for MIMO detection problem. In this algorithm, at the first level of the search tree, P_c metrics corresponding to M parent nodes in M-QAM modulation are computed and sorted in increasing order. K nodes with the smallest metric values are kept and saved in memory and the rest of the nodes are discarded. At the next level, these K nodes become the new parent nodes and are expanded to their respective child nodes. Consequently in this algorithm instead of expanding every node only K selected nodes with the lowest accumulated metrics (PEDs) are expanded. At each level of the tree there are only KP_c new metrics to be computed.

If real-valued decomposition is applied, the number of new metrics will be reduced to KP_r metrics. After completing the tree traversal, only K leaves with the smallest metrics are retained. The leaf with the smallest metric is the detection result. K-best algorithm guarantees a fixed throughput and has BER performance that is close to ML detector. This algorithm requires a distributed buffer structure to store K surviving child nodes and their metrics. In order to find K surviving nodes at each level an efficient sorter is

required. The value of K determines the implementation complexity, throughput, BER performance and latency. In section 5.2, K-best algorithm will be revisited in detail.

3.4.4.4 Sub-optimal sphere detector algorithms

Sphere decoding algorithm can be implemented as both optimal and sub-optimal detector. The main reason for a sub-optimal implementation of algorithm is reducing number of visited nodes which improves throughput performance of the detector. In section 3.4.3.2 it has been discussed that if sphere radius r is too small, ML solution may be eliminated and the overall performance of detector in this case is sub-optimal. Early termination of search and modified norm definition [13] in depth-first sphere decoding also results in sub-optimal behaviour. In section 4.7 a number of performance improving measures for sub-optimal sphere decoding algorithm will be introduced.

3.4.4.5 Fixed sphere decoding (FSD) algorithm

Depth-first sphere decoder discussed in previous section has two main drawbacks:

1. Detector throughput depends on the channel noise level and impairment.
2. Due to sequential and recursive nature of depth-first algorithm, pipelining and parallel processing in hardware are not easily achievable.

Fixed sphere decoding (FSD) algorithm tries to overcome these two limitations, by limiting the search to a fixed number of lattice points **H**s [73]. In this approach the search is limited to a subset of constellation Λ in the modulation scheme. The algorithm performs a fixed tree search on the predetermined paths of the search tree regardless of the channel noise level and impairments. The main advantage of FSD algorithm is that it performs a fixed number of operations during detection process.

As a result of this approach by following fixed predefined paths, unlike the depth-first sphere decoding algorithm, FSD algorithm is non-sequential and can be implemented in an array processing architecture. The independence between the search paths provides parallel search implementations which improves system throughput. An effective FSD detection requires channel matrix ordering which is an added complexity to the detector implementation. The BER performance of FSD algorithm is in low to medium range as specified by Table 3.1. Determining the subset of transmit constellation that needs to be searched and the subsequent channel ordering are the most challenging parts of the FSD

algorithm. In section 4.10, a more detailed discussion of this algorithm along with the VLSI architecture of the algorithm will be presented.

3.5 Soft detection of MIMO signals

In the iterative MIMO receiver, MIMO detector needs to generate a-posteriori probability (APP) about the inner coded bits x_1. The APP is usually expressed as a log-likelihood ratio value (L-value). A decision can be made from an L-value by using its sign to tell whether the bit is a one or zero. The magnitude of an L-value is a measure of the reliability of the decision; L-values near zero correspond to unreliable bits and L-values near one show a reliable decision on the value of the bit. In Appendix A, the basic equations of APP detection have been derived. Full description and proof of equations can be found in [7][8][48].

It is assumed that a block of bits **x** corresponding to one use of the channel model (3-1) is being transmitted from the transmitter. Figure 3.11 shows the block diagram of an iterative MIMO decoder.

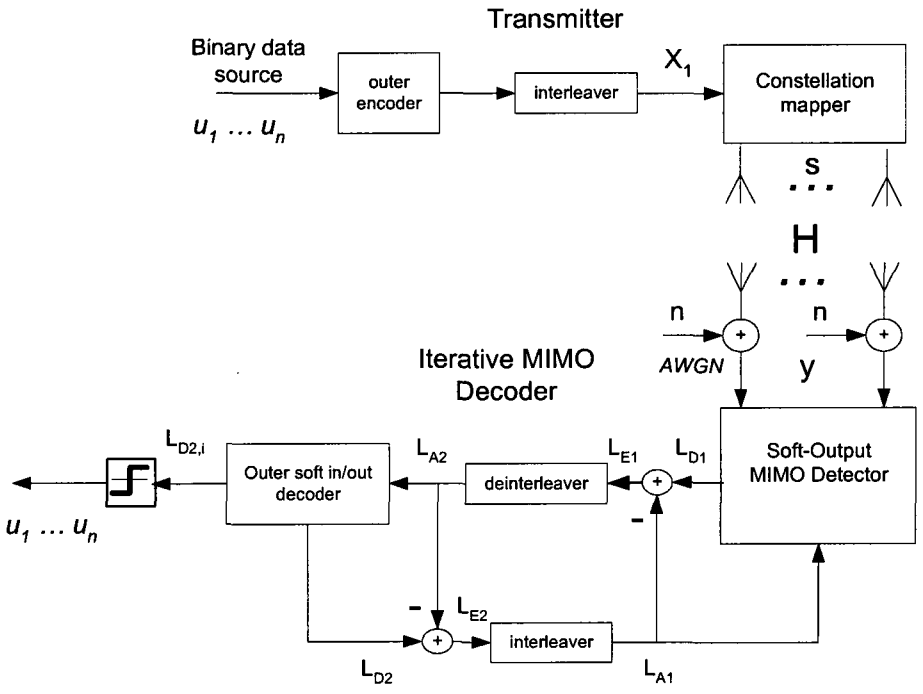


Figure 3.11 Iterative detection for MIMO channels

Even with the approximate computations, $L_E(x_k \mid \mathbf{y})$ is exponential in the length of the bit vector **x** or the number of symbols in the constellation Λ. To find the maximizing

hypothesis in equation (A-17) for each x_k, there are $2^{M.Mc-1}$ hypothesis to search over in each of the two terms. For even a moderate block size M, or bits per symbol M_c this complexity can be very high. For example for a 4x4 16-QAM MIMO system, we have $M.M_c-1 = 15$ and $2^{M.Mc-1}=32768$; for 4x4 64-QAM system we have to search $2^{M.Mc-1}=$ 8388608 hypothesis. Figure 3.12 shows the general block diagram of a soft-output MIMO detector. A short list of select candidates is generated by *symbol list generation* block and is passed on to *soft value computation* block to compute and generate $L_E(x_k \mid \mathbf{y})$ values which are passed to outer soft decoder.

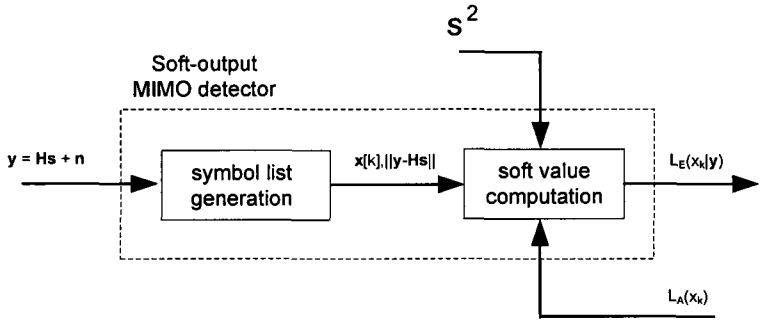


Figure 3.12 Soft-output MIMO detector block diagram

3.5.1 List Sphere Decoding (LSD) Algorithm

List Sphere Decoding (LSD) algorithm is based on sphere decoding (SD) algorithm introduced earlier in section 3.4.3.2. Sphere decoding algorithm can be used to compute $L_E(x_k \mid \mathbf{y})$ in equation (A-17). The modified algorithm was proposed by Hochwald and Ten Brink [7] and is called List Sphere Decoding (LSD) algorithm and has been used widely in the literature. In this algorithm the sphere decoder is modified to generate a list (L) of the N_{cand} point $\mathbf{s}$ that makes $\|\mathbf{y}-\mathbf{Hs}\|^2$ the smallest. This list must include s_{ML} but its size should be sufficiently large to contain the maximizer of equation (A-17) with high probability. To create L, sphere decoder needs to be modified in two ways such that after finding a point inside initial radius that satisfies SC ($\|\mathbf{y}-\mathbf{Hs}\|^2 < r^2$):

1. It should not decrease r to correspond to the radius of this new point.
2. The new point should be added to L if the list is not already full; or if L is full, it should compare this point with the point in L with the largest radius and replace this point if the new point has smaller radius.

The soft information about any given bit x_k is essentially contained in L because if there are many entries in L with x_k=1, then it can be concluded that the likely value for x_k is one and if there are few entries in L with x_k=1, then the likely value is zero. If there are no entries in L with a prescribed bit value, then we can set its corresponding value $L_E(x_k \mid \mathbf{y})$ to an extreme value. A larger r means larger N_{cand} list which makes the list more reliable but it will also cause reduction in the speed of LSD algorithm. Using this algorithm, (A-17) can be approximated as:

$$L_E(x_k \mid \mathbf{y}) \approx \frac{1}{2} \max_{x \in L \cap X_{k,+1}} \left\{ -\frac{1}{\sigma^2} \cdot \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 + \mathbf{x}_{[k]}^T \cdot \mathbf{L}_{A,[k]} \right\} - \frac{1}{2} \max_{x \in L \cap X_{k,-1}} \left\{ -\frac{1}{\sigma^2} \cdot \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 + \mathbf{x}_{[k]}^T \cdot \mathbf{L}_{A,[k]} \right\}$$

(3-19)

Finding N_{cand} points is generally slower than just finding s_{ML} (N_{cand} = 1), because the search radius always stays at r and does not decreases with every point that is found.

For LSD soft-value calculations; it is possible to compute candidate list L per block channel symbol **y** just once and store the data in memory. After one iteration, the updated prior knowledge L_{A1} from the outer decoder is used for calculation of (3-19), by searching the same list L to find the maximizing hypotheses. The choice of r is an important issue in this algorithm that can affect both speed and accuracy of the algorithm. If r is chosen too small, only a few points will be found inside the sphere. On the other hand, choosing r too large slows down the algorithm because it searches through many candidates. Hochwald and Ten Brink [7], proposed an approximate value for r, which is dependent to the channel noise level.

3.5.2 K-best Schnorr-Euchner (KSE) with soft outputs

This algorithm is similar to K-best algorithm which has already been introduced in section 3.4.4.3. In KSE algorithm, the K best paths retained at the last iteration are the candidate list. The hard output would be the path with lowest metrics. The candidate list obtained in KSE does not necessarily include the ML point and consequently performance of KSE algorithm is lower than performance of LSD algorithm.

KSE has a number of advantages in hardware implementations compared with LSD algorithm. KSE is a single-direction searching algorithm, i.e. the search proceeds in the forward direction only and consequently KSE has a fixed throughput. LSD on the other

hand is a two-directional searching algorithm, i.e. the search proceeds forward and backward and as a result its detection throughput is variable [16]. It is possible to improve BER performance of KSE algorithm by considering the second term in equation (3-19) for candidate list reduction [34] but this comes with added computational complexity. Similar to other list type algorithms, the number of survivor paths is a key factor to efficient implementation of KSE algorithm. A larger K would increase the candidate list size and as result adds to the overall complexity of implementation. The main benefit of a large K is in improving BER performance of the detector.

3.6 MIMO detector hardware architectures

3.6.1 Sphere decoding architectures

A folded one node per cycle architecture for depth-first sphere decoding was introduced by Burg *et al.* in [13]. This architecture is composed of two main blocks that operate in parallel to ensure that a new node of the search tree is visited in each cycle. *Metric Computation Unit* (*MCU*) handles forward iterations by identifying the preferred child of the current parent node. *Metric Enumeration Unit* (*MEU*) follows *MCU* on its path through the tree with one cycle delay, to prepare for the moment when the forward iteration stalls because a leaf is reached or sphere constraint (SC) is violated by all children of a parent node. In this case, *MEU* immediately supplies a new node with the associated PED, from which the metric computation can continue in the next cycle.

Two different implementations of this architecture have been demonstrated in [13]. Both implementations operate directly on complex-valued constellations and the implementations mainly differ in the preprocessing block and in realization of SE enumeration. ASIC-I applies exhaustive search enumeration where ASIC-II uses polar enumeration with early termination to improve average decoding delay.

Garett *et al.* proposed a 4x4 16-QAM soft-output spherical decoder [14]. The estimated throughput of the system is 38.8 Mbits/s and estimated area is 10 mm^2 in 0.18 μm CMOS process. The *searcher* unit in this architecture has a latency of two clock cycles and evaluates the sphere radius equation (3-12) and generates a set of valid candidates for calculating APP values.

Since this architecture only processes one search node at a time, one valid candidate is passed back to the head of the pipeline, while the remaining candidates are pushed into the stack. Once *searcher* hits a node with no valid candidates, it retrieves a partial search node from the stack and begins processing that branch. When the candidate search reaches the leaf node of the search tree, it passes that candidate list onto the ML-APP unit which stores the ML solution and builds the APP information.

The *searcher* continues to evaluate the candidates until it completes searching the tree, or it hits the search candidate limit dictated by the throughput requirements. It can generate up to sixteen simultaneous valid candidates. In order to keep the pipeline from stalling; the architecture needs a 15-port memory since one result is always fed back to the beginning of the pipeline. The actual number of valid search candidates generated is a dynamic behaviour of the algorithm and is a function of sphere radius. The X-to-Y interface takes the valid candidates and multiplexes them down to the ports. When the value of X is less than or equal to Y, the stack can store all of the products in a single clock cycle otherwise the pipeline stalls. The authors estimated the complexity of this architecture based on their previous work and reported a throughput of 38.4 Mbits/s and area of 10 mm^2 for a 4x4 16-QAM spherical detector.

Lin *et al.* [66] proposed a reduced complexity sphere detector. In polar child node enumeration [13], candidates are searched by finding the intersection of the constellation circles and the boundary radius as will be explained later in section 4.3. The efficiency of the search has been improved by using a hierarchical radius table and shifting the center of the constellation space. The BER performance of this detector is worse than the sphere detector. The average throughput of this detector is 95 Mbps with 64-QAM modulation.

3.6.2 K-Best architectures

K-best algorithm traverses the search tree breadth-first, i.e. the detector visits all the siblings of a node before it proceeds to the next level. The detection effort is constant over each vector symbol; i.e. the detector visits only K nodes on each level of the tree and computes the metrics of all their child nodes and among these child nodes it selects the nodes with the K smallest metrics as the new parent nodes for the next level.

The choice of design parameter K determines trade-offs between silicon area, throughput, and BER performance. The bit-error-rate (BER) performance improves with increasing K

at the cost of added silicon complexity. The allocation of computing resources and hence computation cycles per vector symbol are constant and determined solely by the design parameter K. The main advantage of K-best algorithm is its fixed throughput.

A pipelined linear array architecture for implementation of K-best algorithm was originally introduced by Wong *et al.* in [12] and was later adopted by others for various implementation of the same algorithm [18][19]. The general practice for implementation of K-best algorithm is to decompose the N_t-dimensional complex-valued equations into a $2N_t$-dimensional real-valued problem using RVD method previously discussed in section 3.4.2. This architecture is composed of a linear array of processing elements (PEs), which are associated with the levels of the search tree as shown in Figure 3.13. The buffers between PEs are used to store the temporary information passed between the processing elements. A K-best decoder which operates directly on complex-valued constellations requires N_t processing element blocks, while an architecture that uses RVD employs $2N_t$ processing elements. In each step, the i-th PE receives a list of K admissible nodes from the preceding $(i+1)$-th PE. The entries of the list contain the partial candidate vector symbols $\mathbf{P}_{i+1}$ of the admissible nodes and the associated metrics. The task of each processing element (PE) is to visit these parent nodes and to identify a set of K preferred children which are then passed on to the next processing element.

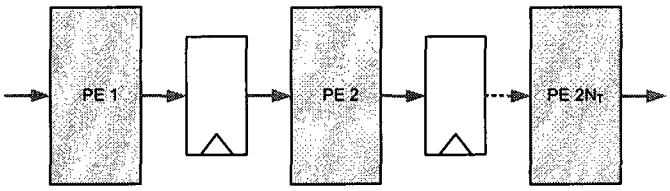


Figure 3.13 K-best pipelined architecture

In [12], Wong *et al.* introduced a fully serial architecture in which the metric computation takes one cycle per child node. The architecture was designed and synthesized for a 4x4 16-QAM MIMO system using CMOS 0.35μm technology and was able to achieve a throughput of 10 Mbits/s. The bubble sort circuit in this architecture consists of K compare-and-select units and $(2K-1)$ registers, after KP_r (P_r is the real value constellation size, as an example for 16-QAM constellation the real valued constellation points are {-3,-1,1,3} and $P_r = 4$) clock cycles the smallest K metric values will be selected by the

bubble sort block. The total number of clock cycles in this case is equal to $(2P_r+2\ N_t)K$ [12].

Wenk *et al.* proposed a new K-best architecture that takes a more parallel approach in design of processing element [36]. This architecture was able to achieve 424 Mbps. In the proposed architecture each processing element consists of a *metric computation unit*, a *K-best unit* that determines the K smallest metrics, and a *register bank* where the K smallest nodes of the previous layer are stored.

Guo and Nilsson proposed a soft-output architecture based on K-best Schnorr-Euchner (KSE) decoding algorithm [16]. The proposed architecture was implemented for a 4x4 16-QAM MIMO system in 0.13 μm CMOS technology and is reported be able to achieve decoding throughput of more than 100 Mb/s with a 0.56 mm^2 core area and 97K gates.

The algorithm used in this architecture is modified KSE and is called MKSE by authors. MKSE algorithm uses the information contained in discarded paths that are not extended to the end sublattice. In other words, MKSE generates the soft outputs by using the discarded paths as well as the K survivor paths during some iterations of the algorithm.

The core of this architecture has already been proposed as a K-best hard decoder [15] and there have been minor modifications to include changes in algorithm and adding an extra module to calculate the soft output data. The soft output module uses the discarded paths from the fourth stage to the last stage and calculates the soft outputs of the MIMO decoder directly. The major issue with this architecture is that it cannot support any iteration with the outer decoder and as result the BER performance of this decoder is lower than iterative decoders. The soft output module calculates the soft values first based on the discarded paths from the fourth stage and the final soft outputs are calculated just after 20 paths are obtained at the last stage. Since the pipeline for computation of soft outputs operates in parallel to the main pipeline, the latency and throughput is not affected by adding the soft-output module to the original hard output decoder.

Complex detector architecture for a 4x4 64-QAM relaxed K-best detector has been proposed by Chen *et al.* in [65] based on a low complexity breadth-first algorithm. In order to reduce the complexity of sorting algorithm, instead of strictly sorting all the metrics at each level for selection of the K best metrics, the approximate sorting has been

used to reduce hardware complexity of the sorter block. The basic idea of the approximate sorting is to break sorting into a number of parallel sorters. Each parallel sorter compares the path metrics with a fixed threshold values. The entire range of path metrics is divided into a number of regions. The path metrics that fall in the same region are not sorted. The main issue with the approximate sorting is that the number of path metrics that fall into each region is not predetermined and as a result throughput of this type of detector unlike other breadth-first detectors is variable. Moreover the buffer sizes should be selected large enough to save all the possible path metrics that pass the approximate sorting and selection.

3.6.3 Depth-first vs. K-Best

Depth-first tree search algorithms are implemented in a sequential, non-pipelined architecture, whereas K-best algorithm is based on a parallel, heavily pipelined hardware structure [12][15][16]. The main advantage of K-best approach is constant throughput, but the drawback is the loss in BER performance and the increase in silicon area and power consumption. The use of radius reduction in sphere decoding algorithm increases the average throughput of depth-first algorithm to the level of K-best algorithms. The throughput of depth-first tree search algorithms drops under poor channel conditions at low SNR values.

3.6.4 Other MIMO detectors

Exhaustive search ML and V-BLAST ASICs have also been reported in the literature [22], [21], [20]. Exhaustive search ML based architectures is only practical for low rate systems where R< 8 bpcu (bits per channel use). The V-BLAST algorithm is mostly used for higher order modulation, as decoding complexity is almost independent of the constellation size but the BER performance of the algorithm is rather poor due its inability to exploit the full diversity available in the channel.

Huang *et al.* proposed a configurable architecture supporting QPSK, 16-QAM and 64-QAM modulation modes over a configurable range of antenna configurations (2x2 to 6x4) [70]. The proposed MIMO detector is divided into two detectors, i.e. a core detector and a residual detector. The core detector is a 2x2 V-BLAST ML detector used for detection of the first two symbols of the incoming received symbols. The residual detector is an ordered successive cancelation (OSIC) detector that detects the rest of the

received symbols. In order to improve throughput of the proposed detector, a variant of the architecture has also been implemented (ASIC II) by adding a parallel search in V-ML core detector and also by using pipeline stages between functional blocks. Simulation results show that the BER performance of this detector is between ML and V-BLAST detectors and outperforms the OSIC detector.

Chapter 4 Depth-first Tree Search MIMO Detectors

4.1 Introduction

In this chapter VLSI implementation of depth-first tree search algorithms for MIMO detection will be explored. After a brief review of depth-first sphere decoding algorithm, a configurable MIMO detector based on depth-first sphere decoding (DFSD) algorithm will be proposed.

The DFSD detector is a multimode architecture that supports BPSK, QPSK and 16-QAM modulation modes and configurable antenna configurations. Due to the exponential increase in complexity of depth-first detector with respect to the number of constellations in M-QAM modulation, the depth-first sphere decoding algorithm in its classical form cannot be expanded to higher order modulation schemes. In section 4.6.1 limitations and challenges of classical sphere detectors are explored and later in section 4.7 a number of algorithmic performance improvement measures are introduced which can be utilized to increase throughput performance of depth-first sphere decoding algorithm.

The modified algorithm is named depth-first-K-best (DFKB) algorithm to underline the dual characteristics of this algorithm in pruning nodes of a search tree. In DFKB algorithm, K child nodes of each parent node are selected for further processing. Throughput simulation results show that the proposed modified algorithm has higher throughput compared to the classical version of sphere decoding algorithm.

The main challenge in VLSI implementation of the DFKB algorithm is enumeration and selection of K preferred child nodes of each parent node. In section 4.8 a low complexity enumeration algorithm and architecture is introduced that provides the main building block for implementation of DFKB algorithm. The proposed enumeration architecture has low hardware complexity and provides the flexibility to support BPSK, QPSK, 16-QAM and 64-QAM modulation schemes.

Metric Enumeration and Computation Unit (*MECU*) is a low complexity processing element that enumerates children of the input parent nodes. It also computes the updated metrics for each of the child nodes. *Metric Enumeration and Computation Unit* (MECU)

is integrated into MIMO detector architecture for implementation of DFKB algorithm. The implementation results for this architecture are presented in section 4.9.

The *MECU* architecture presented in section 4.8.2 can also be used as the basic processing element for implementation of other tree search based algorithms. In section 4.10, an array processing architecture for fixed sphere decoding algorithm based on *MECU* block has been proposed.

Due to iterative nature of depth-first search algorithms, parallel implementation of such algorithms tends to be challenging and requires modifications to the algorithm and its hardware architecture. In section 4.11, the *MECU* based detector architecture has been expanded to include two parallel detectors. In the final section of this chapter implementation results are presented and compared and final concluding remarks are drawn.

4.2 Depth-First Sphere Decoding Algorithm

The Sphere Decoding algorithm (SD) can achieve optimal (ML) bit-error-rate (BER) performance with reduced computational complexity compared to exhaustive search ML detector. In this type of detector the search is limited only to those vector symbols **s** for which **Hs** lies inside a sphere with radius r around the received vector point **y**. We can express this condition by the following inequality:

$$d(\mathbf{s}) = \|\mathbf{y} - \mathbf{Hs}\|^2 < r^2 \qquad \textbf{(4-1)}$$

All possible combinations of transmitted symbols in a MIMO channel can be mapped onto a tree graph. The root of this tree is on level N_t+1 and each node on level i corresponds to the transmitted symbols from N_t to i-th antenna. The leaves on the first level show all the possible combinations of transmitted symbols.

Depth-first search starts at the root of the tree ($i = N_t + 1$) and progresses by expanding one of child nodes of the search tree and progressing down the tree until it reaches the leaves at the first level of the search tree, or it can not progress any further due to a failed test. If a failed test prohibits forward progress, the search backtracks and returns to the most recent node that has not been expanded.

In a depth-first sphere detector, the search tree is traversed depth-first such that at each node after calculating the metric T_i the sphere radius test (SR) is performed. The result of this test determines the direction of the search, i.e. forward or backward traverse.

It is possible to improve the throughput of the sphere detector by shrinking the sphere radius dynamically. The basic idea here is that once a valid solution has been identified, the value of the sphere radius is updated to the node metric value (PED) associated with the new valid solution and the tree search continues with the new radius value. Schnorr-Euchner (SE) ordering rule can further optimize the tree search by giving priority at each level of the tree to the nodes with the smallest metrics [4].

The depth-first sphere decoding algorithm with dynamic radius reduction is outlined in Figure 4.1. Partial symbol vector $\mathbf{P}_i(c_m)$ shows the path over the search tree from root to node c_m and $T_i(c_m)$ is the PED metric associated with $\mathbf{P}_i(c_m)$ such that

$$\mathbf{P}_i(c_m) = [c_m, s_{i+1}, \ldots, s_{N_t}]^T \qquad \textbf{(4-2)}$$

$$T_i(c_m) = T_i(\mathbf{P}_i(c_m)) \qquad \textbf{(4-3)}$$

$$c_m \in \Lambda$$

In equations (4-2) and (4-3), c_m is any constellation point in M-QAM constellation space Λ. The following equations are used in section 3.4.3 to formulate sphere decoding algorithm and are used in Figure 4.1

$$\mathbf{y}_q = \mathbf{Q}^{\mathbf{H}}\mathbf{y} \qquad \textbf{(4-4)}$$

$$\mathbf{H} = \mathbf{QR}$$

$$T_i(\mathbf{P}_i) = T_{i+1}(\mathbf{P}_{i+1}) + \left|e_i(\mathbf{P}_i)\right|^2$$

$$e_i(\mathbf{P}_i) = y_{qi} - \sum_{j=i}^{N_t} R_{ij} s_j$$

In equation (4-4) $\mathbf{Q}$ is an orthogonal and $\mathbf{R}$ is an upper triangular matrix.

Throughput of depth-first sphere detector is inversely proportional to the number of iterations through the algorithm and depends on the initial choice of the sphere radius r_0 as well as channel noise level and parameters. It should be noted that if the initial sphere radius r_0 is too small the sphere constraint may not include the ML solution. The initial choice for r_0 depends on the number of bits used for data presentation. In an 8-bit processor r_0 should be set to the maximum possible number in an 8-bit computing system which is 255.

1. Read channel parameters y_{qi}, R_{ij}, initial sphere radius r_0
2. Set $i=N_t$, $r=r_0$, $T_{i+1}(c_m)=0$, $\mathbf{P}_{i+1}=[]$, $l_k=0$ $k=1,2,..,4$
3. Generate $\mathbf{P}_i(c_m)=[c_m, \mathbf{P}_{i+1}]$, $iteration=iteration+1$
4. Calculate $T_i(\mathbf{P}_i(c_m))$ for all children nodes using (4-4)
5. If there is at least one m : $T_i(\mathbf{P}_i(c_m))<r$ (***SR test***)
   $l_i=length\{ T_i(\mathbf{P}_i(c_m))| T_i(\mathbf{P}_i(c_m))<r\}$
   $T_{i,t}(\mathbf{P}_{i,t}(c_m))=sort(\{ T_i(\mathbf{P}_i(c_m))| T_i(\mathbf{P}_i(c_m))<r\})$ $t=l_i,\ldots,1$
   Save $\mathbf{P}_{i,t}(c_m)$, $T_{i,t}(\mathbf{P}_{i,t}(c_m))$
   Go to step 6
   Else
   Set $l_i=0$, $i=i+1$
   Go to step 7
6. If $i=1$
   Update SR: $r=\min(T_{i,t})=T_{1,li}$
   Save temporary search result: $T_s=T_{1,li}$, $\mathbf{P}_s=\mathbf{P}_{1,li}$
   Go to step 7
   Else
   Set $T_{min}=T_{i,li}$, $\mathbf{P}_{min}=\mathbf{P}_{i,li}$
   Set $l_i=l_i-1$, $i=i-1$
   Set $T_{i+1}=T_{min}$, $\mathbf{P}_{i+1}=\mathbf{P}_{min}$
   Go to step 3
7. If $l_k=0$ $k=1,2,..,4$ (all the branches are pruned)
   Go to step 9
   Else
   $i=i+1$
   Go to step 8
8. If $l_i=0$
   Go to step 7
   Else
   Set $l_i=l_i-1$,
   If $T_{i+1,li}<r$ (***SR test***),
   Go to step 3
   Else
   Go to step 8
9. Search result: T_s, $\mathbf{P}_s$

Figure 4.1 Sphere decoding algorithm

4.3 Enumeration

Metric Enumeration is defined as the process of identifying and listing the child nodes of a parent node that satisfy sphere radius condition. According to Schnorr-Euchner (SE) ordering rule; the listing of child nodes is based on the ascending order of their metric values. In MIMO detection, the metric is *Partial Euclidian Distance* PED (T_i) as defined by equation (4-4). The enumeration algorithms that have been widely used in detection architectures are divided into the following main categories:

Lattice decoding: In the case of sphere decoding with real-valued constellation points, all the admissible children of a parent node can be located within an admissible interval. The boundaries of the interval can be calculated explicitly [5]. The constellation point that is the closest to the centre of that interval has the smallest metric and would be considered as the starting point for SE enumeration. The ordering of the child nodes proceeds in a zigzag fashion to the boundaries of the predetermined interval [4]. It should be mentioned that this method is only applicable to real-valued constellation points and its application to complex valued constellation points requires applying real-valued decomposition (RVD) of complex MIMO model as discussed in section 3.4.2. Figure 4.2 shows lattice enumeration of real-valued constellation points.

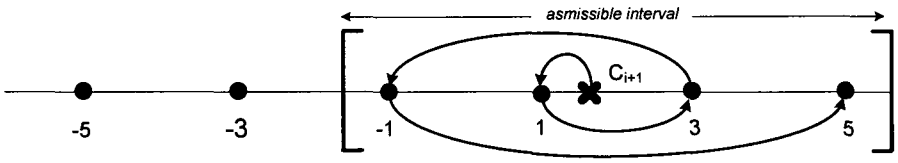


Figure 4.2 Lattice decoding enumeration

Exhaustive enumeration: In this method all the PED values are explicitly computed and are compared with sphere radius. The PED values that pass the sphere radius test are then sorted. The child nodes are processed according to the order of the sorted PEDs. The main advantage of exhaustive approach is its expandability to any complex constellation structure.

Polar partition: This method can be directly applied to complex M-QAM constellation and has been proposed by Hochwald and ten Brink [7] and later modified for VLSI implementation by Burg [13]. The M-QAM complex constellation space is partitioned into P concentric circles centred at the origin. As an example in case of 16-QAM modulation there are 3 concentric circles that partition the entire constellation space. The boundaries of admissible intervals are defined as the intersection of concentric circles and the circle centred at C_{i+1} (equation (3-11)) with sphere radius r as shown in Figure 4.3.

The process of finding the closest point on each concentric circle can be further simplified for hardware realization by locating C_{i+1} in a polar partitioned space. The PEDs of preferred children from each of the P subsets are compared to find the smallest

PED. It should be mentioned that polar enumeration for M-QAM only finds the smallest PED and the enumeration of other points are performed by exhaustive search.

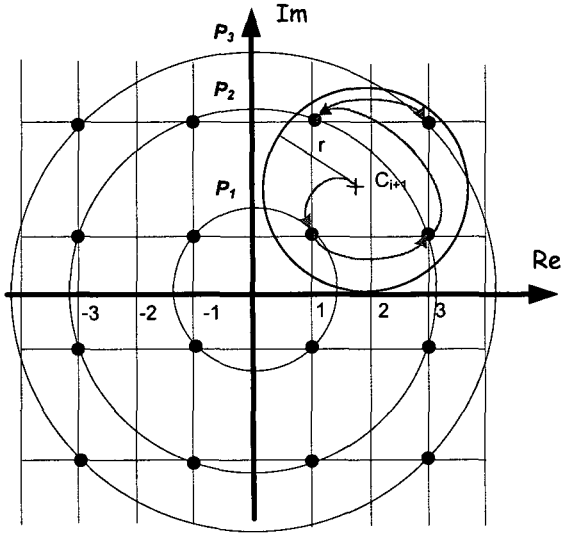


Figure 4.3 Polar partition enumeration (16-QAM modulation)

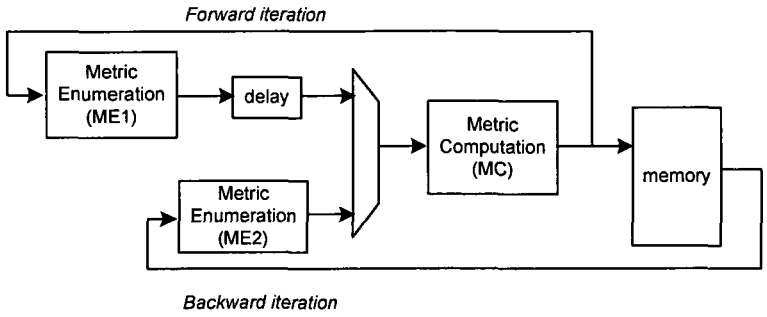
Cartesian partition: The general approach in enumeration based on Cartesian partition is to divide the M-QAM constellation space into a number of partitions and find the location of C_{i+1} within this partitioned space. In section 4.8.1, an enumeration algorithm based on Cartesian partition will be introduced.

4.4 Folded architecture

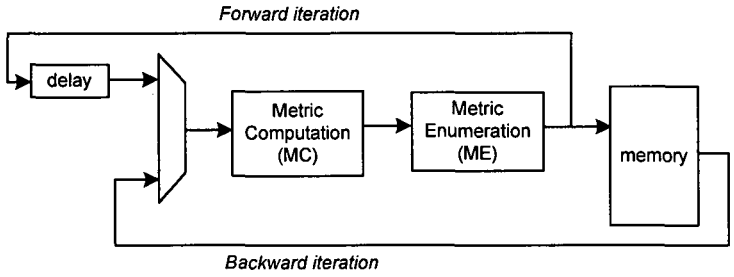
Depth-first sphere decoding algorithm is a recursive algorithm which can be implemented in hardware using folded architectures [53]. The block diagrams of two possible configurations of the folded architecture are shown in Figure 4.4. The *Metric Computation (MC)* block handles the forward iterations by computing the metrics for child nodes of the current parent node. The *Metric Enumeration (ME)* block enumerates the child nodes and selects the node that should be considered as the next parent node, this can happen in both forward traverse mode and also when the forward iteration stalls because a leaf is reached or sphere constraint is violated by all the child nodes.

In *Architecture I*, the enumeration process has been divided between the two metric enumeration blocks *ME1* and *ME2*. In a forward traverse mode, *ME1* selects the next child node while *ME2* selects the next node from the child nodes saved in memory.

Architecture II, on the other hand uses one enumeration block that enumerates all the child nodes before storing them in the memory.



(a) Architecture I



(b) Architecture II

Figure 4.4 Folded architecture configurations

An example of the tree traverse using folded architecture is shown in Figure 4.5. In this example during the first three cycles, the search proceeds in forward direction, feeding the output of *MC* back to its inputs (i.e. the preferred child of the previous cycle becomes the parent node in the current cycle). *ME* chooses nodes B1 and C1 as the preferred children of nodes A1 and B1 respectively. In the third cycle, the forward iteration stops and the sphere radius is updated by the *MC* based on the node metric of C1.

In the example shown in Figure 4.5, *ME* has already computed the metrics of node B2 and it determined that the metric of node B2 had violated the sphere radius constraint. A violation of sphere constraint leads to the elimination of B2 from the list of preferred children and consequently node A2 is selected as the new parent node to be visited by the MC in the fourth cycle.

It should be noted that the backward iteration to A2 also happens on the third cycle and the decoder can immediately continue the forward direction traverse to node D1 and finally to the leaf E1 on the fourth and fifth cycles.

On the fifth cycle, once again sphere radius is updated and the search continues until all the possible paths in the search tree are either traced or eliminated. The path to the final traced leaf is the search result and the solution to the detection problem.

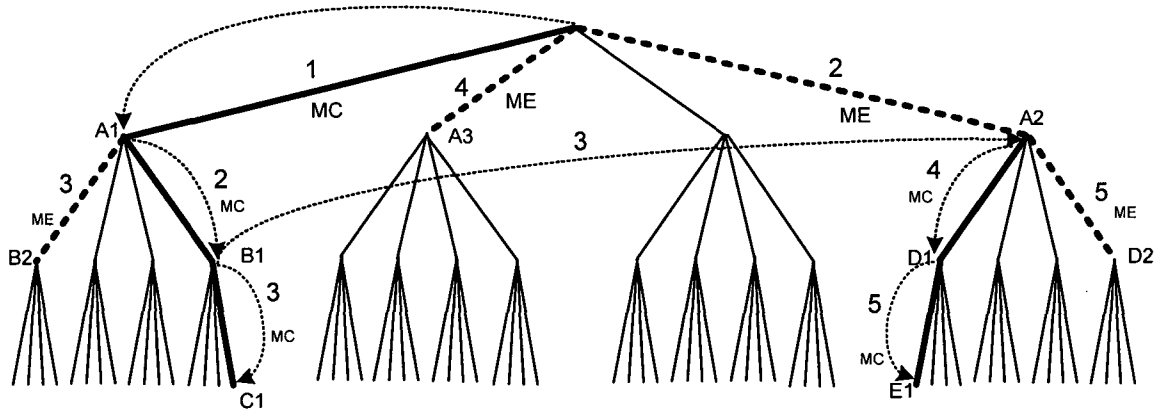


Figure 4.5 Tree pruning steps in one-node-per-cycle architecture

4.5 Configurable depth-first sphere detector architecture

In this section a configurable architecture for depth-first sphere detector based on exhaustive enumeration is proposed. The introduced MIMO detector supports 16-QAM, QPSK and BPSK modulation and two to four transmitting antennas. Previously in Figure 4.4 two possible configurations for depth-first MIMO detectors have been introduced. The detector architecture introduced in this section follows the configuration presented in Figure 4.4(a) with two *Metric Enumeration* (ME) blocks on forward and backward iteration paths processing one node of search tree on every clock cycle.

4.5.1 Search algorithm

The flow chart of the configurable sphere decoding algorithm is shown in Figure 4.6. The algorithm has two main branches; *node processing* and *buffer control.*

During an iteration of the algorithm, *node processing* branch receives the relevant information of a parent node from buffers and expands it to all of its child nodes. This information includes metric T_{i+1} and the partial symbol vector of the node $\mathbf{P}_{i+1}$. The new metrics are calculated for each of the child nodes of the selected parent node and the results are compared to the sphere radius r.

Vector V_r shows the result of this comparison for each of the child nodes. The mask vector m defines the validity of each of the child nodes based on the type of modulation

scheme selected for that tree level. Vector V_i is the final validity vector of child nodes and is stored in buffers along with updated metrics and partial symbol vectors.

The *buffer control* branch of flow chart determines the search direction. In the forward traverse mode, the search continues with symbols that pass the sphere radius test until the search reaches the first level of search tree ($i = 1$).

After each sphere radius test, the minimum value of the metrics that pass the test are found and the child node associated with the minimum metric is directly sent to *node processing* branch. The rest of metrics along with their associated partial symbol vectors ($\mathbf{P}_i$) are stored in the buffers assigned to level i (buf_i). If no metrics pass the sphere radius test or the algorithm reaches the first level of the tree ($i = 1$) a *pop* operation occurs and a parent node from previous level of the tree is selected to continue the forward traverse of the tree. The search stops when all the buffers (buf_m, $m=2,N_t$) are empty and at this point $\mathbf{P}_i$ shows the search result.

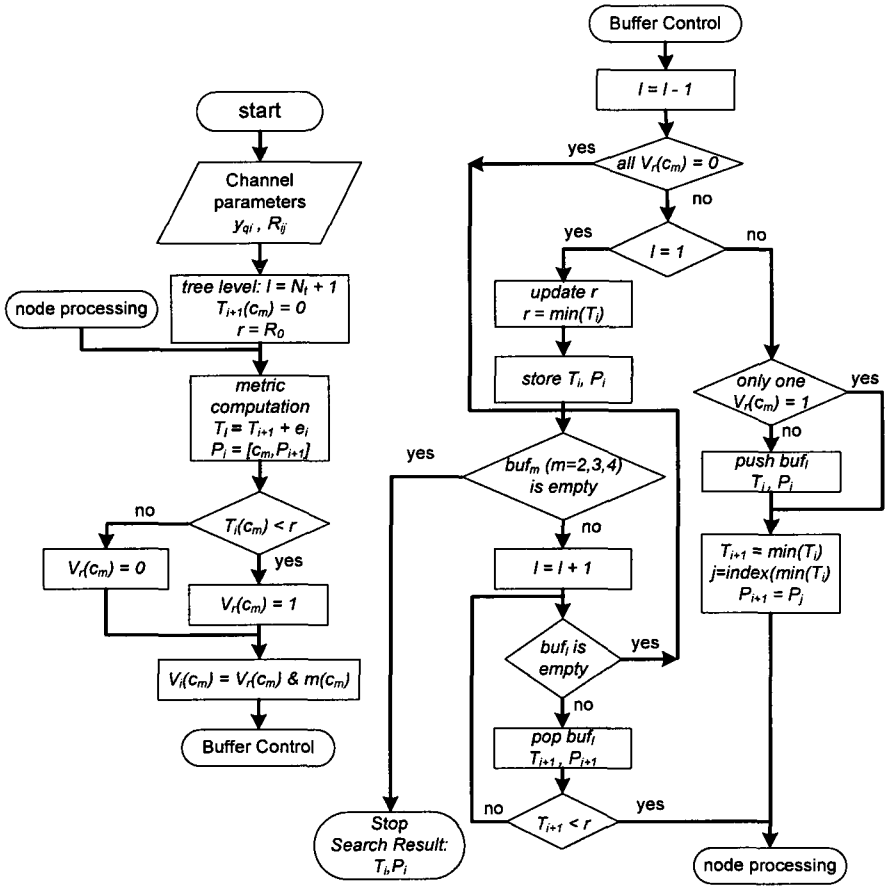


Figure 4.6 Data flow chart of the proposed sphere decoding algorithm

In order to better clarify the sphere decoding algorithm, the forward and backward movements on the search tree has been further explained using two examples demonstrated in Figure 4.7. The search tree on the left side of the figure shows the first four iterations in which the search starts at level 4 of the tree and continues until a temporary result is found at level 1. At each level, one of the child nodes is selected for further processing and the rest are either stored in the buffers assigned for that level or discarded. If none of the child nodes pass the sphere radius test; a saved node from higher levels is retrieved from buffers. The search tree on the right side the figure shows the fifth iteration in which c_3 is retrieved from buf_2 and passed on to *node processing* branch.

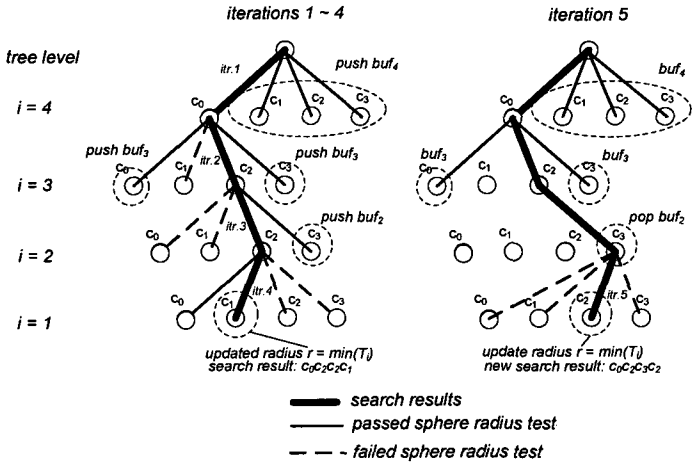


Figure 4.7 Forward and backward movement in depth-first tree search

The operation of *buffer control* branch of the algorithm is similar to stack algorithm used in searching tree graphs in software. Figure 4.8 shows contents of stack during forward (*push*) and backward (*pop*) steps (**c1** to **c8**) along the search tree. The buffer in Figure 4.8 has been divided into three submodules associated with different tree levels.

The tree search starts at level 4 (step **c1**) by computing the metrics for all the nodes on this level. Node $\mathbf{s_0}$ is directed to the processing pipeline (*node processing* branch) for further expansion to its child nodes and $\mathbf{s_1}$, $\mathbf{s_2}$ and $\mathbf{s_3}$ are saved in the third buffer module. The search moves forward by expanding $\mathbf{s_0}$ in step **c2**. In step $\mathbf{s_2}$ is selected for passing to pipeline for expansion; $\mathbf{s_0}$ and $\mathbf{s_3}$ are saved in the second buffer module; and $\mathbf{s_1}$ fails the sphere radius test and is dropped from the search.

The search moves forward in steps **c3** and **c4** to the first level by further expanding the tree. In step **c4**, $\mathbf{s_3}$ is popped out from the first buffer and is expanded to its child nodes.

The search continues in both directions by visiting saved nodes in buffers and eventually stops when there are no more nodes left in the buffer modules.

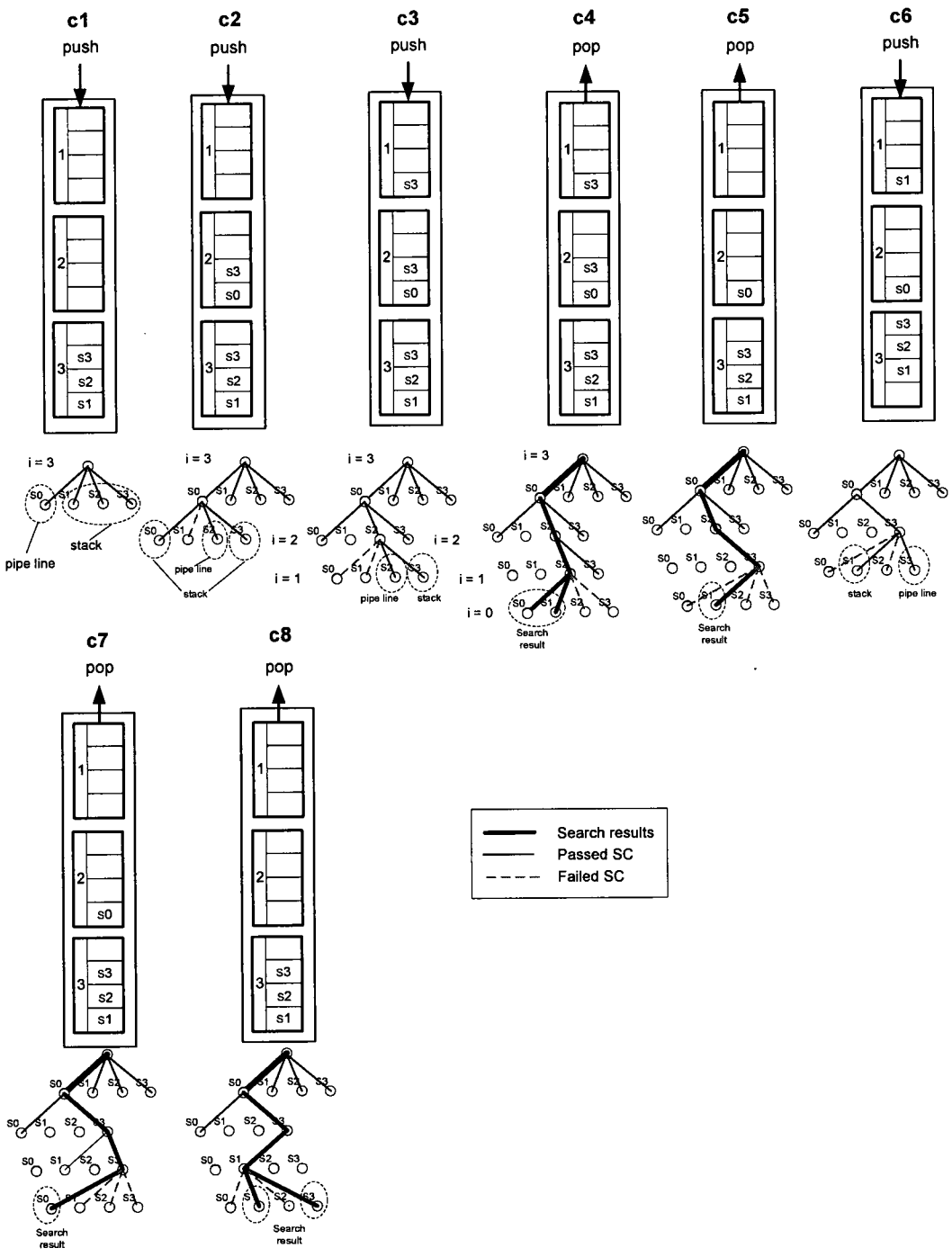


Figure 4.8 Stack operation

Due to modular structure of stack architecture; it can be easily reconfigured to support different number of antennas. Each sub-module in stacked buffer is associated with one

level of tree; if the number of antennas changes one can simply disable the extra sub-modules in the stack buffer. The key to efficient implementation of this architecture is design of a multi-port stack memory that can support simultaneous write operation on all its ports. Within the stack modules, SE ordering can be implemented using buffer reordering techniques and in parallel to main data path.

4.5.2 Detector Architecture

The sphere decoding algorithm introduced in section 4.5.1 had two main branches that work in parallel to reduce the processing cycles of the detector. The sphere decoding algorithm can be efficiently implemented in hardware using folded architectures as previously discussed in Figure 4.4. In this section a configurable depth-first sphere detector architecture based on *Architecture I* in Figure 4.4 will be presented. As previously stated in section 4.4, in this configuration there are two enumeration blocks that select the next node that should be processed.

The top level architecture of the proposed detector has been shown in Figure 4.9, depicting the main blocks in the design [58]. The received symbol vector $\mathbf{y}$ after preprocessing is mapped to $\mathbf{y}_q$ such that

$$\begin{aligned} \mathbf{y}_q &= \mathbf{Q}^{\mathbf{H}}\mathbf{y} \\ \mathbf{H} &= \mathbf{QR} \end{aligned} \tag{4-5}$$

Where $\mathbf{H}$ is the channel matrix, $\mathbf{Q}$ is an orthogonal and $\mathbf{R}$ is an upper triangular matrix.

In every clock cycle, one of the parent nodes is expanded to all its child nodes; this includes calculating the associated metrics for all children, comparing the metrics with the sphere radius, selecting the nodes that pass the sphere radius test and applying the modulation mask to filter out the unwanted nodes.

The received symbols (y_{qi}) enter the *Metric Computation* block along with matrix $\mathbf{R}$, T_{i+1}, $\mathbf{P}_{i+1}$ and *level* signal. The task of *Metric Computation* block is to update the PED metrics T_i and generate the new set of partial symbol vectors $\mathbf{P}_i(c_m)$ as defined by equations (4-2), (4-3) and (4-6).

$$\begin{aligned} T_i(\mathbf{P}_i) &= T_{i+1}(\mathbf{P}_{i+1}) + \left|e_i(\mathbf{P}_i)\right|^2 \\ e_i(\mathbf{P}_i) &= y_{qi} - \sum_{j=i}^{N_t} R_{ij} s_j \end{aligned} \tag{4-6}$$

PED metrics are compared to sphere radius and saved in *Data Buffer* block. Multiple data vectors can be saved in *Data Buffer* simultaneously.

In the next processing cycle, T_{i+1} and $\mathbf{P}_{i+1}$ are selected from *Min_T_i* or *Data Buffer*. *Control Unit* block controls multiplexers M_1 and M_2 and decides on direction of the search. In a forward traverse move on the search tree the nodes are enumerated in *Min_T_i* block and the node with the smallest metric is selected to be the new parent node in the next cycle.

In case of backward moving to higher tree levels, the next parent node is popped out of *Data Buffer* block which enumerates the saved nodes and selects the next parent node with the smallest metric. The search stops when *Data Buffer* is empty and the search is at level one. The search result $\mathbf{P}_{res}=[s_1, s_2, s_3, s_4]$ is saved in *symbol buffer*.

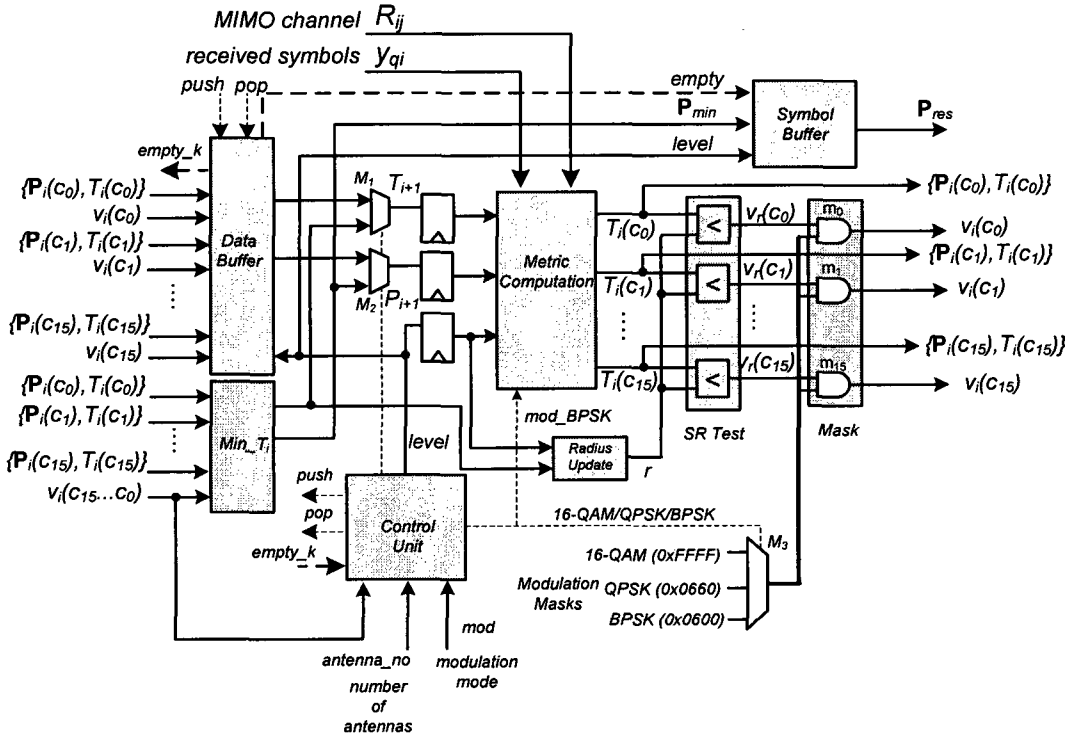


Figure 4.9 Multi-mode sphere detector architecture

Metric Computation

Metric Computation block computes PEDs of all the child nodes ($T_i(c_0)$, $T_i(c_1)$, ..., $T_i(c_{15})$) of a parent node in parallel considering 16-QAM constellation based on equation (4-9). Figure 4.10 depicts the architecture of the metric computation block. The constellation for 16-QAM modulation has been identified with (c_0,c_1, ...,c_{15}). In Figure

4.10, *level* signal selects one of the rows of matrix **R** (R_{ij}) to be used in equation (4-6). The selected symbols from previous levels (s_2, s_3, s_4) are identified by the partial symbol vector $\mathbf{P}_{i+1}$.

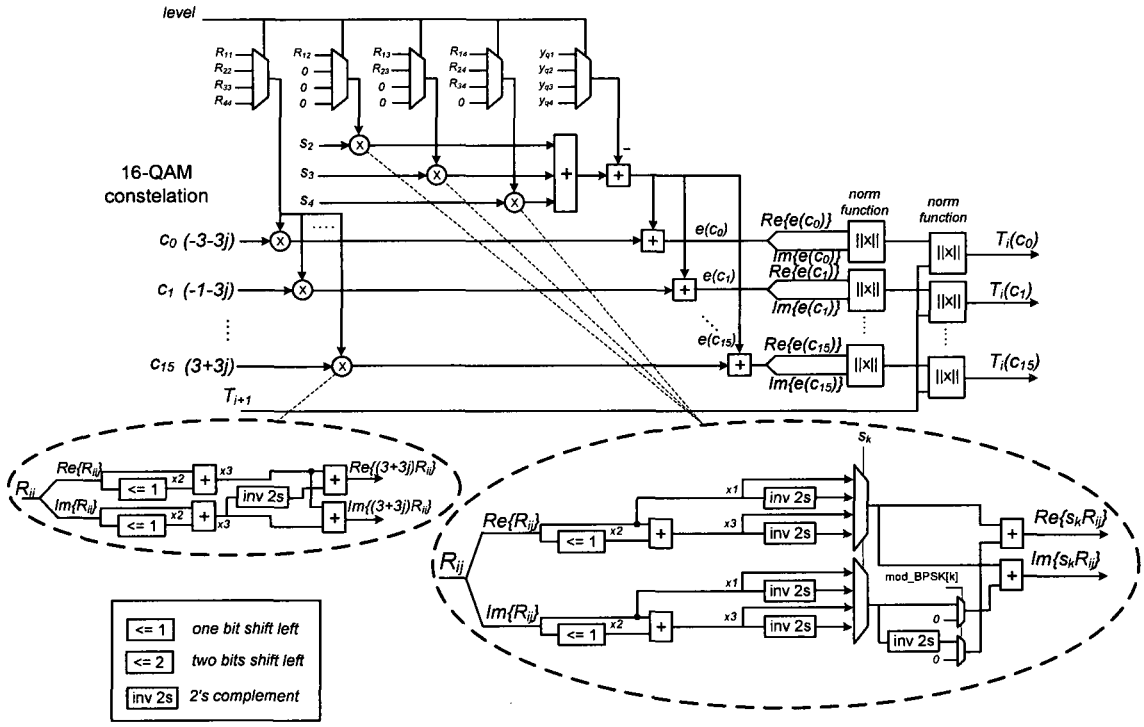


Figure 4.10 Metric computation block

All complex multipliers have been implemented using shift-add operations in order to minimize hardware complexity of the multiplication operations. In Figure 4.10 and Figure 4.11, implementations of a number of complex multipliers have been shown. The mod_BPSK vector identifies tree levels in which symbols that are modulated with BPSK modulation are located. In case of BPSK modulation constellation c_9 and c_{10} are used in complex multiplication and the imaginary parts of products are forced to zero as shown in Figure 4.11.

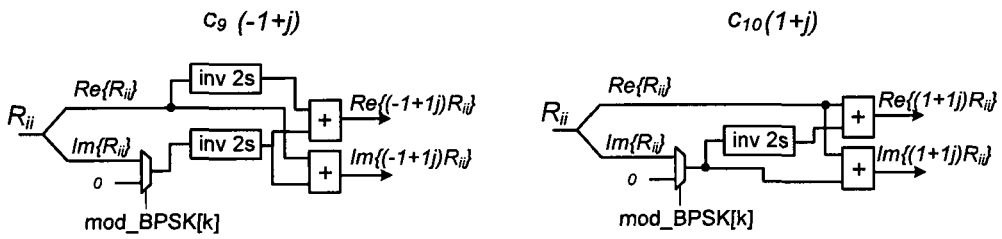


Figure 4.11 Complex multiplication- c_9 c_{10}

SR Test

The role of the parallel comparators in Figure 4.9 is to compare the value of the updated node metrics $T_i(c_m)$ with the sphere radius (r). The resulting vector $V_r(c_m)$ shows the validity of each of the child nodes.

Mask

The constellation points in QPSK modulation can be considered as a subset of 16-QAM modulation. Figure 4.12 shows the constellation space for 16-QAM, QPSK and BPSK modulations. *Metric Computation* block computes the metrics $T_i(c_m)$ considering 16-QAM modulation for all tree levels, i.e. $c_m \in \Lambda_{16\text{-}QAM}$.

As already discussed (Figure 4.11) in *Metric Computation* block, BPSK modulation has been considered as a special case of QPSK modulation by modifying complex multiplications for c_0 and c_1 constellations. In case of QPSK and BPSK modulation modes the metrics of constellations which are not part of QPSK or BSPK constellations should be discarded. The modulation masks selected by the multiplexer M_3 in Figure 4.9, sets the valid bits associated with the unwanted constellations to zero. *Modulation mask* is defined as a 16-bit binary number $0xb_{15}\ldots b_2b_1b_0$ where b_m corresponds to one of the constellations in Figure 4.12. If b_m is set 0, the corresponding metric $T_i(c_m)$ and partial symbol vector $\mathbf{P}_i(c_m)$ should be discarded.

Using a series of parallel *AND* gates in *Mask* block the selected modulation mask is applied to vector V_r. The resulting validity vector $V_i(c_m)$ shows the validity of each constellation and is saved in *Data Buffer* along with metric value $T_i(c_m)$ and partial symbol vector $\mathbf{P}_i(c_m)$ as shown in Figure 4.9.

The modulation type for each of the transmitting antennas is selected by modulation mode (*mod*) input. At each level of search, *Control Unit* selects the appropriate *modulation mask* to reflect the modulation scheme for that tree level.

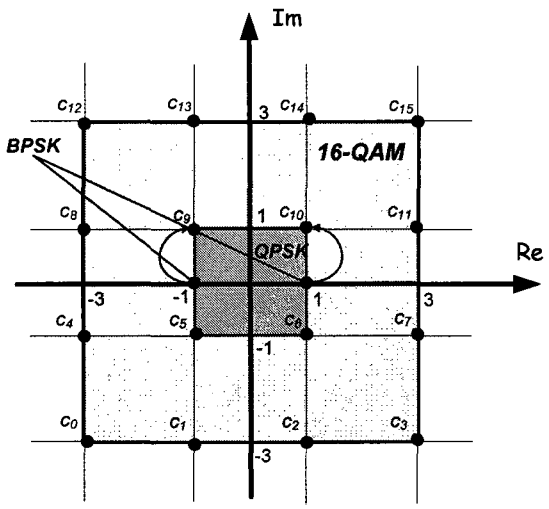


Figure 4.12 16-QAM/QPSK/BPSK constellation

Min_T_i

In forward movement on the search tree, the node with the smallest metric has to be selected as the new parent for the next cycle of processing. *Min_T_i* block in Figure 4.9 compares all the generated metrics $T_i(c_m)$ and finds the node with the smallest metric. *Min_T_i* functionality is similar to *ME1* enumeration block in *Architecture I* in Figure 4.4

Radius Update

Radius Update block in Figure 4.9 dynamically updates the sphere radius at level one of the search tree such that

$$r = \min(T_1(\mathbf{P}_1(c_m)) \qquad \textbf{(4-7)}$$

Where $\min(T_1(\mathbf{P}_1(c_m)))$ is the minimum value of all PEDs at level one generated by *Min_Ti* block.

Data Buffer

Figure 4.13 shows the architecture of *Data Buffer* block, which consists of three buffer modules (*BUF_4*, *BUF_3*, *BUF_2*) assigned to levels 4, 3, and 2 of the search tree. *Buffer Control* selects the buffer modules based on tree level and number of transmitting antennas. It also controls the write and read operations in each buffer module. Write operation to all the locations within each module is simultaneous.

Each module enumerates the valid saved nodes and selects the next entry that should be read based on SE ordering rule (section 4.3). The internal buffer enumeration performs the same functionality as *ME2* enumeration block in *Architecture I* in Figure 4.4. The

modular structure of *Data Buffer* permits independent disabling of *BUF2* and BUF_3 modules in order to be able to support 3x3 and 2x2 antenna configurations.

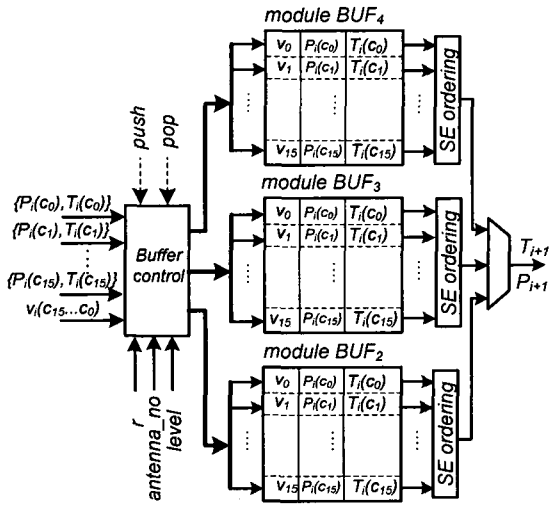


Figure 4.13 *Data buffer* architecture

In Figure 4.14 the internal architecture of *Data Buffer* modules has been depicted. As stated before, each module stores all the information related to a node, i.e. metric value $T_i(c_m)$, partial symbol vector $\mathbf{P}_i(c_m)$, and valid bit $V_i(c_m)$.

A set of parallel comparators in *SR test* block (Figure 4.14) continuously compare the metrics for each of the valid nodes with the updated sphere radius and set the corresponding valid bits to zero if the comparison fails. This process is in parallel to the main data path and increases the overall throughput of the detector since the metrics that fail to satisfy sphere radius test are eliminated internally before entering the main processing pipeline.

SE ordering rule has been implemented by searching for the minimum valid metric in *min_indx* block. This metric and its associated symbol vector will be popped out of the buffer module on the next read from *Data Buffer* and its valid bit will be set to zero.

The status of valid bits shows the validity of saved metrics $T_i(c_m)$ and partial symbol vectors $\mathbf{P}_i(c_m)$ in the buffer module. Valid bits can be updated from three different sources as shown in Figure 4.14, i.e. valid bits generated by the *Mask* block in Figure 4.9, updated valid bits from *SE ordering-Enumeration* unit within each buffer module and updated valid bits from *SR test* unit. A set of multiplexers select the source of update.

If all the valid bits in a buffer module are set to '0', the *empty_k* signal is asserted which shows that the buffer module is empty. If all the buffer modules are empty; *Data Buffer* assert *empty* signal which shows that the search is over and detector is ready to process new symbols.

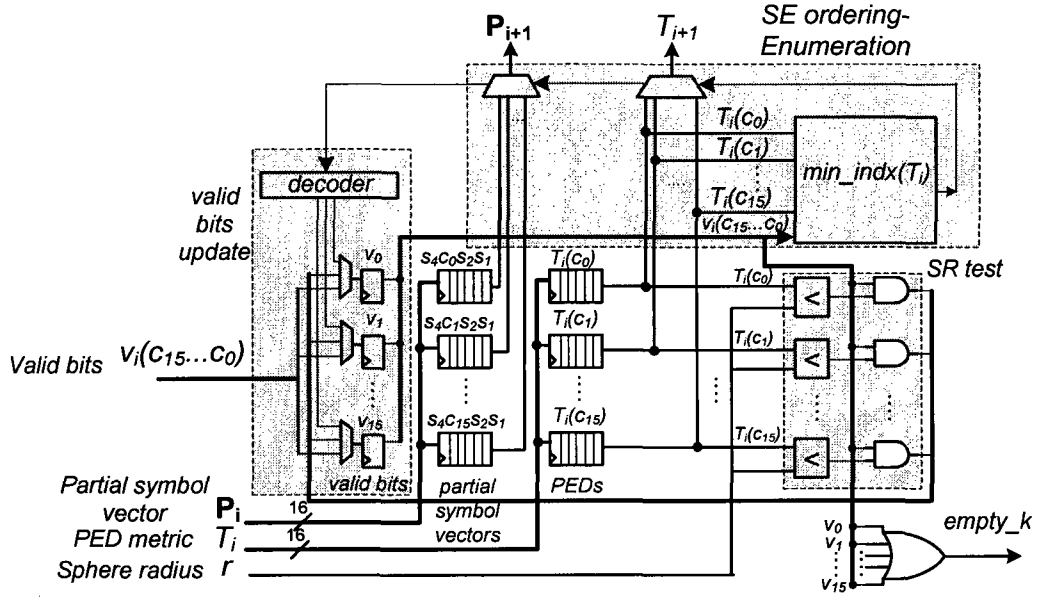


Figure 4.14 Internal architecture of buffer modules

Control Unit

Control Unit block is the control center of the detector architecture shown in Figure 4.9. The main tasks of *Control Unit* can be listed as follows:

- Detection start/stop control
- Forward/Backward search control
- *Data Buffer* Write/Read control
- Detector reconfiguration (modulation mode and antenna configuration)

The *empty_k* signals show the status of each of the buffer modules (BUF_k) inside *Data Buffer* block. The configuration signals *mod* and *antenna_no* reconfigure *Control Unit* for different modulation modes and antenna configurations. The output *level* signal determines next level of the search tree and also selects the buffer module BUF_k in *Data Buffer* block. Write and read operations in BUF_k are controlled with *push* and *pop* signals.

The main state machine in *Control Unit* is *Buffer Control* which has been depicted in Figure 4.15. This state machine controls the forward and backward movements of the search as well as *push* and *pop* operations. States LEVEL4, LEVEL3, LEVEL2 are associated with different levels of search tree and each controls data storage in one of the buffer modules BUF_4, BUF_3 and BUF_2.

The search direction in each state depends on the outcome of sphere radius tests generated by the *Mask* block as well as the current state in the state machine. Transition to a lower level state with more that one valid $V_i(c_m)$ causes a *push* operation in the buffer associated with the destination state which also implies a forward transition over the search tree. If there is only one valid $V_i(c_m)$, there will still be a transition to lower level states but *push* signal will not be asserted. In this situation, the valid metric will be directly used on the next clock cycle to further expand the tree search and there will be no write to the buffer module associated with the state and the buffer remains empty.

In a forward transition mode, the outputs of *Min_Ti* block are selected as T_{i+1} and P_{i+1} by M_1 and M_2 multiplexers as shown in Figure 4.9.

If there are no valid sphere radius test results (all $V_r(c_m)=0$), a *pop* signal will be asserted and the state machine either stays at the current state or moves up to a higher state level. In both cases a new parent node's information (T_{i+1} and $\mathbf{P}_{i+1}$) are read from the appropriate submodule BUF_k in *Data Buffer* block. In this case the outputs of *Data Buffer* block are selected as T_{i+1} and P_{i+1} by M_1 and M_2 multiplexers.

Buffer Control state machine has the capability to be configured for different antenna configurations. The number of transmit antennas determines the next state following the RESET state. As an example in a 3x3 antenna configuration, the first state following RESET state is LEVEL3. This means that under this antenna configuration, BUF_4 will not be used.

Architecture of *Data Buffer* and *Control Unit* state machines can be expanded to support more than four antennas by adding more states each representing one added antenna and repeating the state machine structure as presented in Figure 4.15.

The modulation modes for tree levels are defined by *mod* signal. *Control Unit* asserts *mod_BPSK* signal whenever the modulation of symbols at the tree level identified by

level signal is BPSK. *Control Unit* also controls M_3 multiplexer for 16-QAM and QPSK configurations as previously discussed in *Mask* block subsection.

The detection process stops when *empty_k* signals for all the buffer modules are asserted signalling readiness of detector to accept a new set of received symbols for processing.

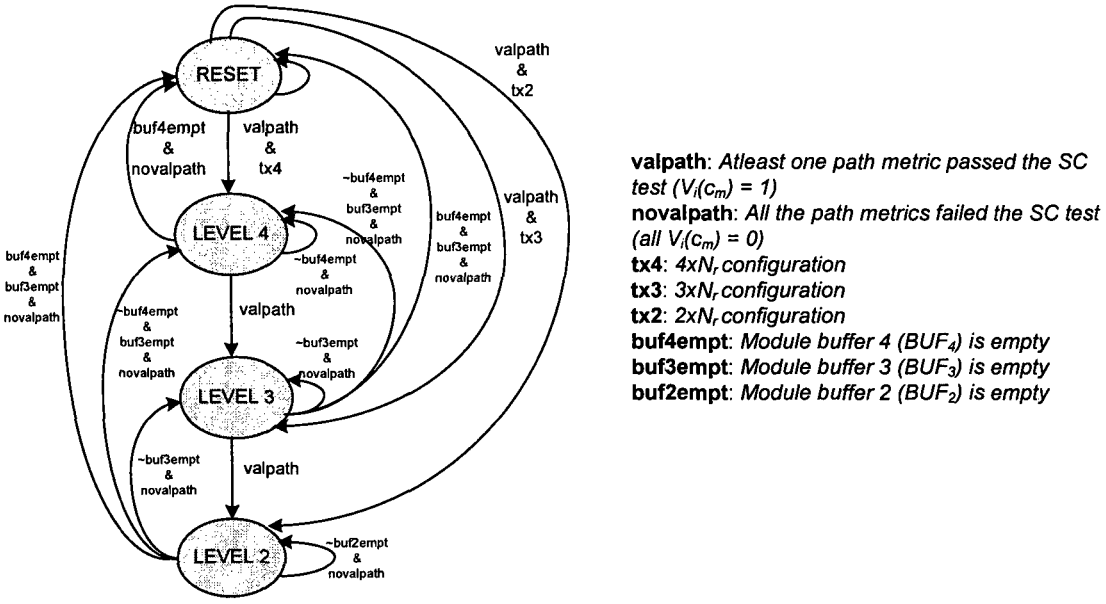


Figure 4.15 *Buffer Control* state diagram-*Control Unit*

4.5.3 Performance and complexity analysis

A 2×2/3×3/4×4 (BPSK/QPSK/16-QAM) MIMO detector has been designed based on the architecture shown in Figure 4.9. After synthesis, timing closure and mapping to 0.13 μm CMOS standard cell library, a clock frequency of f_{clk} = 120 MHz has been achieved with a logic complexity of 65K equivalent gates (equivalent gates number is calculated by dividing total area by the area of a two-input drive-1 NAND gate). All the arithmetic computations have been implemented based on 16-bit fixed point numbers. Data buffers have been implemented using registers to increase the processing speed of the detector at the expense of silicon area.

Figure 4.16 depicts the bit-error-rate (BER) performance of the detector under different operating modes. The average throughput of the detector is inversely proportional to the average symbol processing delay as indicated in equation (3-13). In Figure 4.17 the symbol processing delay of the detector in cycles as a function of signal to noise ratio (SNR) for 4×4 16-QAM mode has been shown.

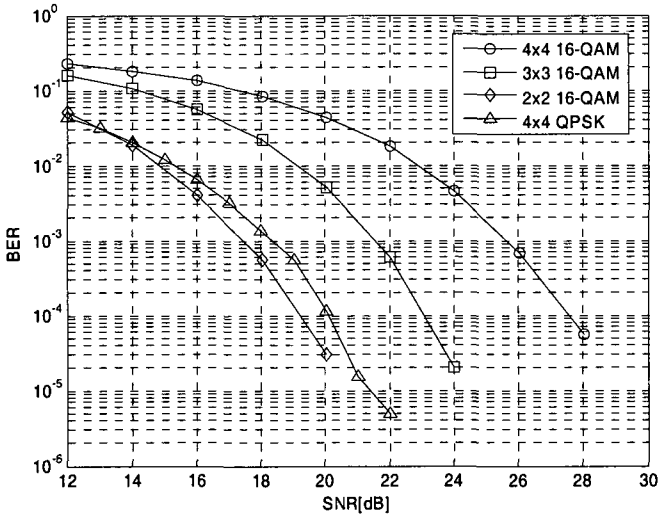


Figure 4.16 BER performance of depth-first sphere detector

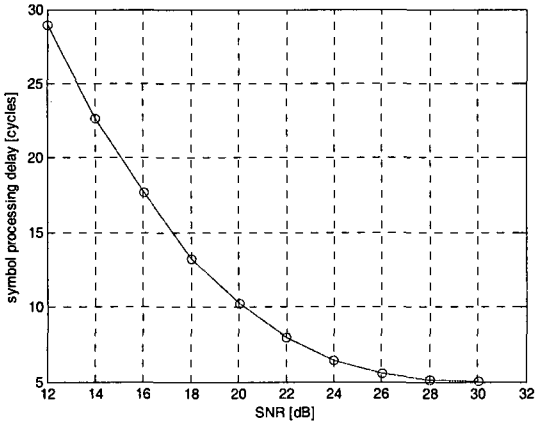


Figure 4.17 Sphere detector symbol processing delay (4x4 16-QAM mode)

In Table 4.1 the performance figures of the proposed detector architecture have been presented and compared with two reported sphere detector architectures in [13]. The throughput figures have been compared for 4×4 16-QAM configuration. One of the main differences between ASIC-I and ASIC-II is applying modified norm definition in ASIC-II that increases the throughput at the expense of a loss in BER performance. It is possible to improve the throughput of our proposed sphere detector by using l^{∞}-norm definition and reduce the number of symbol processing delay. This strategy will be further discussed in section 4.7.1. It should be mentioned that there is an overhead in area and timing related to the added hardware flexibility in DFSD architecture.

Table 4.1 Comparison of SD implementation results for SD algorithm

Reference	ASIC-I [13]	ASIC-II [13]	This work (DFSD)
Number of Antennas	4×4	4×4	4×4, 3×3, 2×2
Modulation	16-QAM	16-QAM	16-QAM, QPSK, BPSK
Detector	sphere detector	sub-optimal sphere detector	sphere detector
Tech [μm]	0.25 μm	0.25 μm	0.13 μm
Equivalent gate number	117 K	50 K	65 K
Clock Frequency	51 MHz	71 MHz	120 MHz
Throughput @ 20 dB	73 Mbps	169 Mbps	213 Mbps (4×4 16-QAM)
Max. Throughput in 4x4 mode	-	-	384 Mbps (16-QAM) 192 Mbps (QPSK)

4.6 Sphere decoding for 64-QAM modulation

The depth-first sphere decoding algorithm introduced in theory can be applied to any M-QAM modulated signal stream. In practice we observe a number of challenges and limitations as the modulation moves beyond 16-QAM. In this section a number of algorithm modifications are proposed that can improve the performance of sphere decoding algorithm for 64-QAM modulation.

4.6.1 Implementation challenges and limitations

There are two main challenges in applying depth-first sphere decoding algorithm in detection of 64-QAM modulated signals. In 64-QAM modulation, number of required memory cells for saving intermediate results increases significantly compared to 16-QAM modulation. As an example, the memory space required to save the search results in a 16-QAM implementation considering 16-bit fixed-point data representation is equal to 1536 bits. If the same architecture is expanded to support 64-QAM modulation, the required memory space will be increased to 7680 bits which shows 500% increase in the number of used memory cells.

The second issue related to the implementation of sphere decoding algorithm for 64-QAM modulation is throughput. In depth-first sphere decoding, throughput of the algorithm is related to the level of signal to noise ratio (SNR). Number of iterations through algorithm (or number of visited nodes in tree search graph) is a direct measure of

the overall throughput of algorithm. In the sphere decoding algorithm outlined in section 4.2, a single iteration of algorithm is considered to be going through steps 3 to 7. In a typical hardware implementation of sphere decoding algorithm; this process may take a number of clock cycles depending on the hardware implementation.

In 64-QAM modulation; number of visited nodes increases significantly at lower SNR values. This is mainly due to the number of nodes that are saved in buffers at early iterations of algorithm when sphere radius is too big. The algorithm revisits all these nodes later which amount to significant increase in the number of iterations [59].

In the following sections we introduce two modifications to the original algorithm to address above issues. In section 4.3 a review of enumeration algorithms has been presented. Hardware complexity of sorting based enumeration algorithms increase exponentially as the order of the constellation increases. In section 4.8, a low complexity enumeration algorithm that supports 64-QAM modulation will be introduced.

4.7 Improved performance sphere decoding algorithms

4.7.1 Modified norm definition

The recursive equations presented by equations (4-4) for metric computation (PED) in tree search algorithms can be approximated by replacing l^2-norm with less complex norm definitions. This process reduces both computational complexity of detection algorithm as well as the hardware complexity associated with computation of squares in l^2-norm definition [13]. It will be shown later that by replacing l^2-norm with less complex l^∞-norm the efficiency of sphere decoding algorithm represented by the number of iterations through algorithm will also improve. The downside of applying these approximations is deviation of detector bit-error-rate (BER) performance from ML performance to sub-optimal performance.

Equation (4-8) is the recursive PED computation formula previously derived in section 3.4.3.

$$T_i(\mathbf{P}_i) = T_{i+1}(\mathbf{P}_{i+1}) + \left|e_i(\mathbf{P}_i)\right|^2 \tag{4-8}$$

In equation (4-8), $T_i(\mathbf{P}_i)$ and $T_{i+1}(\mathbf{P}_{i+1})$ can be substituted with their square roots as defined by equation (4-9).

$$t_i = \sqrt{T_i(\mathbf{P}_i)} \qquad (4\text{-}9)$$
$$t_{i+1} = \sqrt{T_{i+1}(\mathbf{P}_{i+1})}$$

Substituting t_i, t_{i+1} in equation (4-8) yields the following expression for metric computation.

$$t_i = \sqrt{t_{i+1}{}^2 + |e_i|^2} \qquad (4\text{-}10)$$

It should be noted that the above expression is the formal definition of Euclidian norm or l^2-norm. From computational accuracy perspective the two expressions in equations (4-8) and (4-10) are equivalent except for the added complexity associated with calculating the extra square operations and the additional square-root operation.

In order to reduce computational complexity, it is possible to approximate the l^2-norm function with less complex norm definitions [77]. The l^1-norm and l^∞-norm definitions have been particularly useful for reducing complexity of sphere decoding algorithm [24]. In l^∞-norm definition, maximum of the operands is being considered as the approximate norm of the operands as stated in equation (4-11).

$$t_i \approx \max(|t_{i+1}|, |e_i|) \qquad (4\text{-}11)$$

In equation (4-11), the term $|e_i|$ represents the norm of complex-valued $e_i(\mathbf{P}_i)$ defined by equation (3-11). Here we can use l^∞-norm approximation again to compute $|e_i|$ from real and imaginary components of e_i. The approximate forms of equations (3-9) and (3-10) have been presented in equation (4-12).

$$T_i^{\infty}(\mathbf{P}_i) = \max(T_{i+1}^{\infty}(\mathbf{P}_{i+1}), |e_i(\mathbf{P}_i)|) \qquad (4\text{-}12)$$
$$|e_i(\mathbf{P}_i)| = \max(|\Re\{e_i(\mathbf{P}_i)\}|, |\Im\{e_i(\mathbf{P}_i)\}|)$$

$T_i^{\infty}(\mathbf{P}_i)$ and $T_{i+1}{}^{\infty}(\mathbf{P}_{i+1})$ in equation (4-12) represent l^∞-norm PEDs. In the rest of this dissertation the terms $T_i(\mathbf{P}_i)$ and $T_{i+1}(\mathbf{P}_{i+1})$ are used to represent PEDs irrespective of the type of norm used for PED computation.

Simulation results for a 4x4 64-QAM MIMO system presented in Figure 4.22 and Figure 4.23 show that using l^∞-norm definition decreases number of iterations compared to l^2-norm definitions by 30% and BER deviation from ML BER performance is 1 dB which is within the range of sub-optimal BER performance for MIMO detectors.

The sphere radius defined by l^∞-norm has faster shrinkage compared to l^2-norm based sphere radius. Assume two detectors; one based on l^∞-norm and the other based on l^2-norm reach the same node on level one; the sphere radius in each case will be updated as:

$$r_{l^2} = \min(T_1^{l^2}(\mathbf{P}_1(c_m)) \qquad \textbf{(4-13)}$$
$$r_{l^\infty} = \min(T_1^{l^\infty}(\mathbf{P}_1(c_m))$$
$$r_{l^\infty} \leq r_{l^2}$$

The smaller sphere radius removes more nodes and reduces number of saved nodes at early stages of the search which means that in average l^∞-norm based detector requires less number of cycles to detect symbols.

4.7.2 Depth-First-K-Best (DFKB) sphere decoding algorithm

One of the challenges in hardware implementation of 64-QAM sphere decoding algorithm is the number of memory cells required to save partial symbol vectors for previous child nodes. In order to limit the number of required memory cells, one solution would be to select only a number (K) of preferred child nodes and discard rest of the nodes before proceeding to the next level of search tree.

The selection criteria is based on the fact that nodes with smaller metrics have higher chances of reaching a valid answer for the search while the nodes with bigger metric values have higher chances of elimination. Early pruning of child nodes not only reduces the memory space required but also improves the overall throughput of the decoder. This new pruning strategy is a departure from strict depth-first strategy to a combination of depth-first and breadth-first search strategy [59]. It should be noted that the node metrics for selected nodes still need to satisfy sphere radius constraint.

An example of limited selection of nodes has been demonstrated in Figure 4.18 which shows a search tree with four levels. In this example, at each level only two of the nodes that pass the sphere radius test are retained and the rest are discarded.

In order to apply above algorithmic modification, step 5 of the original sphere decoding algorithm needs to be changed such that only the first K elements of sorted PEDs are saved and the rest are discarded. The modification in step 5 of sphere decoding algorithm has been outlined in Figure 4.21

Simulation results for a 4x4 64-QAM MIMO system presented in Figure 4.22 and Figure 4.23 show that BER performance loss is less than 1.1 dB compared to the original SD

algorithm with l^2-norm definition but number of iterations has been reduced by 75% at lower SNR values (< 25 dB). For DF2B (K=2) implementation which saves only two child nodes, the required number of memory cells for storing intermediate results will be reduced to 2720 bits which is a 65% reduction in memory usage compared to the original implementation of the algorithm.

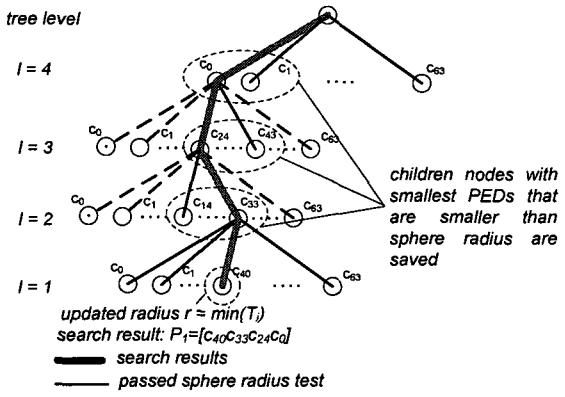


Figure 4.18 Tree traverse in DFKB algorithm

4.7.3 Level-based sphere radius

In order to further reduce number of visited nodes and iterations through the algorithm we need to apply a more effective tree pruning strategy to reduce the number of saved nodes at early stages of the search and higher levels of the search tree. In a tree graph eliminating a parent node removes all the child nodes connecting to it. As a result a smarter method of finding parents nodes that their children will eventually fail the test helps to decrease number of visited nodes or iterations through the algorithm.

In a *forward pruning* strategy, tree expansions that seem to fail are detected and pruned earlier. The real task in applying a forward pruning strategy is to identify which parent nodes are worth considering more closely by expanding to their child nodes and which ones can be pruned off with minimal risk of pruning the ML solution.

Two main approaches can be considered for implementing a more effective pruning strategy in sphere decoding algorithm; i.e. faster shrinkage of sphere radius or a new definition for sphere radius. In section 4.7.1 it has been discussed that by applying l^∞-norm definition, sphere radius shrinks faster which improves tree pruning.

In this section the later approach is followed and a new definition of sphere radius which is a function of tree level will be proposed. In sphere decoding algorithm with dynamic

radius update (Figure 4.1), the sphere radius is being updated at level one of the search tree; such that:

$$r = \min(T_1(\mathbf{P}_1(c_m)) \qquad \textbf{(4-14)}$$

In equation (4-14), $T_1(\mathbf{P}_1(c_m))$ is PED of all the child nodes at level 1. This updated radius will be used to prune branches of tree at every level and remains unchanged until algorithm reaches level 1 of the tree and finds a new result with smaller value. For further simplification, $T_i(c_m)$ and $\mathbf{P}_i(c_m)$ are defined as:

$$\mathbf{P}_i(c_m) = [c_m, s_{i+1}, \ldots, s_{N_t}]^T \qquad \textbf{(4-15)}$$

$$T_i(c_m) = T_i(\mathbf{P}_i(c_m)) \qquad \textbf{(4-16)}$$

In sphere decoding, sphere radius test requires the following condition to be satisfied at every level:

$$T_i(c_m) < r \qquad \textbf{(4-17)}$$

In our proposed approach, at each level of the tree, PEDs are compared with the associated sphere radius for that level. The main idea is that if PED of a parent node is growing faster than the previously detected smallest path, there is a higher risk for the children of that parent node to fail later in the search [59]. As an example for a 4x4 system we have:

$$\mathbf{P}_1 = [c_j, s_2, s_3, s_4]^T \qquad \textbf{(4-18)}$$

$$\mathbf{P}_2 = [s_2, s_3, s_4]^T \qquad \textbf{(4-19)}$$

$$\mathbf{P}_3 = [s_3, s_4]^T \qquad \textbf{(4-20)}$$

In this example, $\mathbf{P}_1$ is the partial symbol vector that results in the minimum metric. Sphere radius can be defined for each level of the search tree as:

$$r_1 = T_1(\mathbf{P}_1) \qquad \textbf{(4-21)}$$

$$r_2 = T_1(\mathbf{P}_1)$$

$$r_3 = T_2(\mathbf{P}_2)$$

$$r_4 = T_3(\mathbf{P}_3)$$

Due to the definition of PED we have the following relationship for the sphere radii defined in (4-21).

$$r_1 \geq r_2 \geq r_3 \geq r_4 \qquad \textbf{(4-22)}$$

In Figure 4.19 the concept of level based radius definition has been illustrated using an example. In this figure the partial symbol vector associated with the temporary search result is $\mathbf{P}_1=[c_{40},c_{33},c_{24},c_0]$ and the four levels of sphere radius (r_1, r_2, r_3 and r_4) are demonstrated graphically.

The effect of defining a sphere radius which is a function of tree level is that at early stages of tree search (l=4,3), PEDs are compared to a smaller number than the original definition of sphere radius and there is a higher possibility of pruning branches at earlier stages.

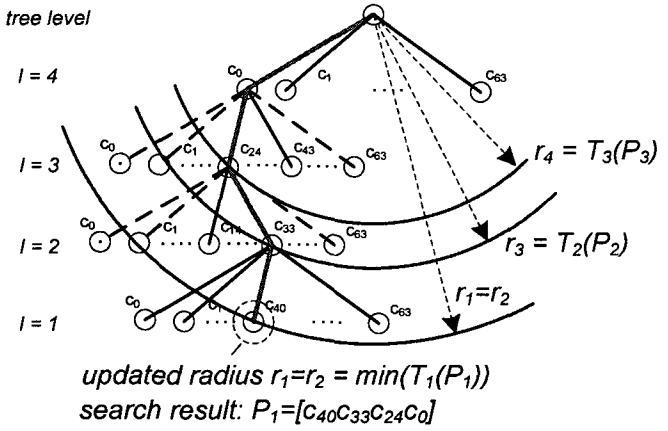


Figure 4.19 Level based sphere radius

Figure 4.20 shows the effect of pruning of parent nodes at level 4 and level 3 of the search tree by plotting the PED values at level 4 ($T_4(c_m)$) and level 3 ($T_3(c_m)$) for all the constellation points in 64-QAM modulation, i.e. $c_m \in \Lambda_{64\text{-}QAM}$. The horizontal lines in the figure shows sphere radius level considering new definitions (r_3, r_4) compared with the original definition of sphere radius (r). Simulation results presented in Figure 4.20 shows that by applying new sphere radius definition at level 4, close to 95% of nodes has been eliminated compared to 37% elimination achieved by original sphere radius definition. At level 3 we observe a 17% improvement in the number of eliminated parent nodes.

Simulation results also show that level based sphere radius is more effective in reducing number of iterations when SNR is smaller than 25 dB. At higher SNR values number of iterations merges to the minimum number of possible iterations which is four. Hence the new sphere radius definition can be applied to smaller SNR values. In practice number of iterations (or processing cycles) can be used as a measure of applying new radius definitions.

In order to implement the above modification, steps 5 to 8 of the sphere decoding algorithm has to change as outlined in Figure 4.21.

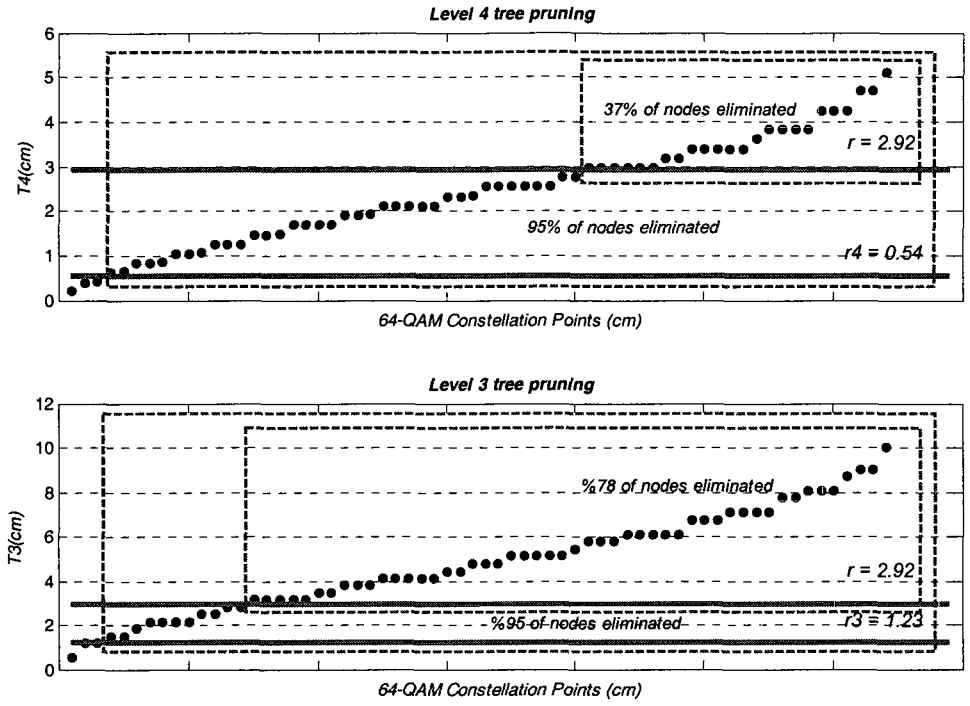


Figure 4.20 Node elimination in level-based sphere radius

4.7.4 Performance simulations

Simulation results for bit-error-ratio (BER) and number of iterations as a function of signal to noise ratio (SNR) for the four algorithms discussed in section 4.7 have been presented for a 4x4 64-QAM MIMO system in Figure 4.22 and Figure 4.23. In a typical hardware implementation of sphere decoding algorithm, the average decoder throughput is inversely proportional to the number of iterations. As the simulation results show, the original sphere decoding algorithm with l^2-norm and l^∞-norm (SD-l2 and SD-linf) has a poor performance (higher number of iterations) at lower SNR values (SNR<25). In DF2B algorithm which saves the two best children of each parent node, number of iterations decreases by 76% at low SNR values due to fewer numbers of saved nodes at earlier stages of the search. At higher SNR values numbers of iterations for all the four cases converge to four which is the minimum number of iterations in a depth-first SD algorithm.

5. If there is at least one m : $T_i(\mathbf{P}_i(c_m))<r_i$ (***SR test***)
   l_i=*length*{ $T_i(\mathbf{P}_i(c_m))|\ T_i(\mathbf{P}_i(c_m))<r$}
   $T_{i,t}(\mathbf{P}_{i,t}(c_m))$=*sort* ({ $T_i(\mathbf{P}_i(c_m))|\ T_i(\mathbf{P}_i(c_m))<r$}) t=$l_i,\ldots,1$
   Save $\mathbf{P}_{i,t}(c_m)$, $T_{i,t}(\mathbf{P}_{i,t}(c_m))$ t=min(l_i,2),1
   Go to step 6

   Else
   Set l_i=0, i = i+1
   Go to step 7

6. If i = 1
   Update SR: If *iteration* <= 5
   r_k=min($T_{i,t}$)=$T_{1,li}(\mathbf{P}_{1,t})$ k=1,2,3,4
   Else
   r_1=r_2=$T_{1,l1}$, r_3=$T_{2,l2}$, r_4=$T_{3,l3}$
   Save temporary search result: T_s=$T_{1,li}$, $\mathbf{P}_s$=$\mathbf{P}_{1,li}$
   Go to step 7

   Else
   Set T_{min}= $T_{i,li}$, $\mathbf{P}_{min}$=$\mathbf{P}_{i.li}$
   Set l_i=l_i-1, i = i-1
   Set T_{i+1}=T_{min}, $\mathbf{P}_{i+1}$=$\mathbf{P}_{min}$
   Go to step 3

7. If l_k=0 k=1,2,..,4 (all the branches are pruned)
   Go to step 9
   Else
   Set i = i+1
   Go to step 8

8. If l_i=0
   Go to step 7
   Else
   Set l_i=l_i-1,
   If $T_{i+1,li}<r_i$ (***SR test***),
   Go to step 3
   Else
   Go to step 8

Figure 4.21 Modified SD algorithm

Figure 4.23 shows BER performance of DF2B sphere decoding with original sphere radius and level-based sphere radius (LBSR) compared to the performance of original SD algorithm with l^2-norm and l^∞-norm. The original SD algorithm with l^2-norm definition (SD-l2) is ML type detector. The BER performance of l^∞-norm and DF2B algorithms shows 1 dB deviation from ML performance. The performance degradation of DF2B with level based sphere radius compared to the ML (SD-l2) detector at 30 dB signal to noise ratio (SNR) is 1.3 dB while compared to SD-linf, the performance degradation is 0.3 dB.

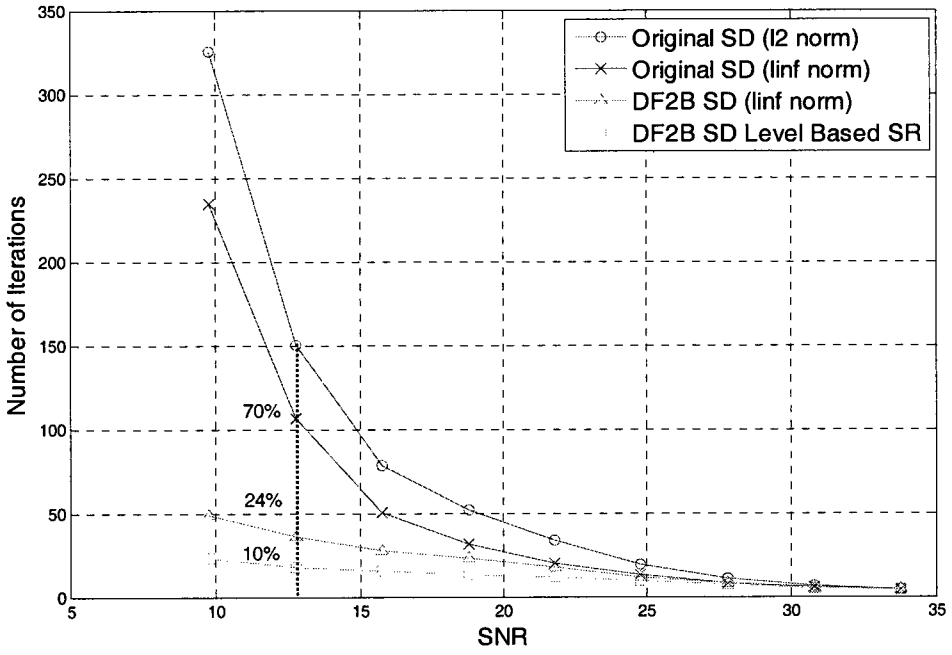


Figure 4.22 Iteration number comparison

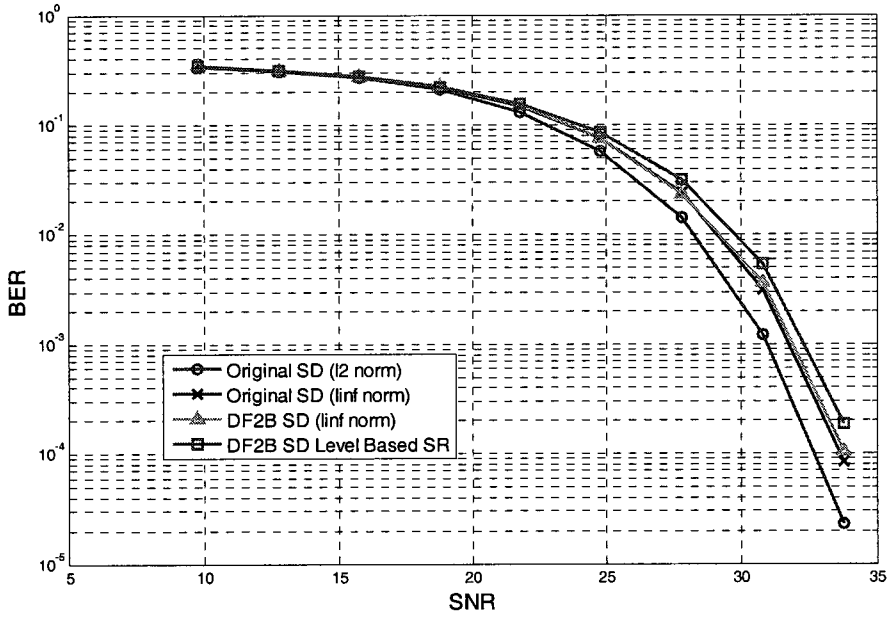


Figure 4.23 BER performance comparison

Sphere decoding algorithm is an effective algorithm for decoding received symbols in MIMO communication at high SNR values. In case of detection of 64-QAM modulated signal streams, there are a number of challenges that should be overcome. In Table 4.2,

performances of different implementations of sphere decoding algorithms have been compared based on key performance merits. DF2B algorithm reduces the number of required memory cells by 75% and increases throughput of the detector at low SNR by 66% compared to sphere decoding (SD) algorithm with l^{∞}-norm. In order to further improve throughput of the algorithm a new definition for sphere radius has been proposed which defines radius as a function of search tree levels. Level-based sphere radius (LBSR) shows 84% improvements in throughput compared to original sphere decoding (SD) algorithm with l^{∞}-norm. Simulation results also show that BER penalty is less than 2 dB compared to original sphere decoding (SD) algorithm with l^{2}-norm. One of the advantages of the proposed modifications is that they do not require modifications to channel preprocessing.

Table 4.2 Performance comparison of SD algorithms

Algorithm	Original SD (l^2-norm)	Original SD (l^{∞}-norm)	DF2B SD (l^{∞}-norm)	DF2B SD LBSR (l^{∞}-norm)
Number of iterations @12 dB	150	106	36	15
BER @30 dB	$1.2\text{x}10^{-3}$	$3.1\text{x}10^{-3}$	$3.6\text{x}10^{-3}$	$5.4\text{x}10^{-3}$
BER deviation from ML curve @30 dB	0 dB	1 dB	1.1 dB	1.3 dB
Number of memory cells (bits)	7680	7680	2720	2720

4.8 Low complexity enumeration algorithm

It has been discussed in section 4.7 that the key to successful implementation of DFKB algorithm is a low complexity enumeration algorithm that can enumerate K child nodes with the smallest accumulated metrics. In this section an enumeration algorithm will be proposed to address this issue. The proposed algorithm covers two other major tasks in sphere decoding, i.e. metric computation and sphere radius test.

The ultimate goal here is to design a processing element which is dynamically configurable and performs three main tasks in tree search based detection i.e. *metric enumeration, metric computation* and *sphere radius test. Metric Enumeration and*

Computation Unit (*MECU*) has been designed as a processing element in tree search based detection algorithms with the ability to perform those three tasks.

Figure 4.24 shows interface signals of the *MECU* processing element. The received symbol y_{qi} and diagonal channel matrix **R** are already defined in equation (4-4). *Modulation mode* parameter configures MECU for processing different types of modulation modes. This parameter can be changed depending on the modulation types of each of the transmitting antennas. The relation between metrics T_{i+1} and T_i is defined in equations (3-8), (3-9) and (3-10). Partial symbol vectors P_{i+1} and P_i identify nodes on tree graph for levels $i+1$ and i respectively.

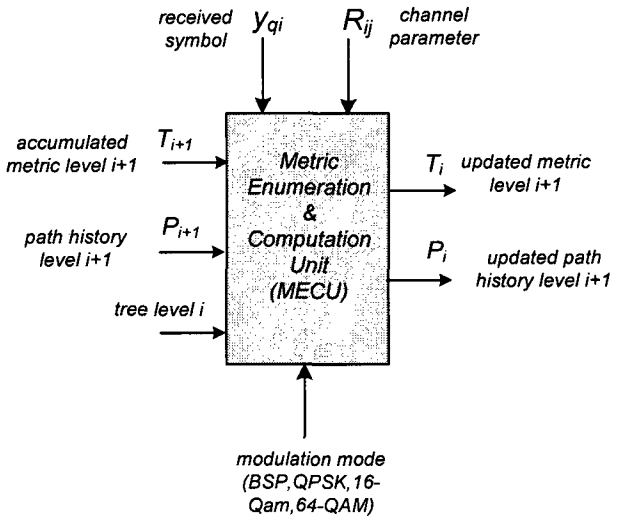


Figure 4.24 MECU block interface

4.8.1 Low Complexity Cartesian Enumeration algorithm (*LCCE*)

In sections 3.4.3.2 and 4.7.1, the recursive sub-optimal solution for sphere decoding has been formulated as:

$$s_{ML} = \arg\min_{s\in\Lambda} \sum_{i=1}^{N_t} \left| y_{qi} - \sum_{j=i}^{N_t} R_{ij} s_j \right|^2 \tag{4-23}$$

$$\begin{aligned} T_i(\mathbf{P}_i) &= \max(T_{i+1}(\mathbf{P}_{i+1}), |e_i(\mathbf{P}_i)|) \\ |e_i(\mathbf{P}_i)| &= \max(|\Re\{e_i(\mathbf{P}_i)\}|, |\Im\{e_i(\mathbf{P}_i)\}|) \end{aligned} \tag{4-24}$$

where

$$T_{N_t+1}(\mathbf{P}_{N_t+1}) = 0 \qquad (4\text{-}25)$$
$$T_i(\mathbf{P}_i) \geq T_{i+1}(\mathbf{P}_{i+1})$$

and

$$e_i(\mathbf{P}_i) = C_{i+1} - R_{ii}s_i \qquad (4\text{-}26)$$
$$C_{i+1} = y_{qi} - \sum_{j=i+1}^{N_t} R_{ij}s_i$$

Where $\mathbf{P}_i$ is the partial symbol vector and s_i is the selected symbol for level i which is above the current tree level. The task of enumeration algorithm proposed in this section is to find nine constellation points in M-QAM constellation space that are closets to C_{i+1} and are within the sphere with radius r, order these constellation points with respect to their metric values and calculate the metric values of each node.

The closest constellation point with the minimum Euclidian distance can be decided by the location of C_{i+1} in the transformed constellation space $\Lambda_R=R_{ii}*\Lambda$. Before proceeding with the details of the algorithm, we assume that Λ_R has been divided into 4M equally spaced partitions. Figure 4.25 shows the first quadrant of Λ_R space along with C_{i+1}. Two different cases have been demonstrated; in Figure 4.25(a), the location of C_{i+1} is within the boundaries of the constellation space (gray shaded square). Figure 4.25(b) shows an example of boundary cases where C_{i+1} is located outside of the constellation grid. The closest constellation to C_{i+1} is the center of gray shaded square and has been named as $\mathbf{u_0}$. Eight closest constellations to C_{i+1} are also shown in the figure.

The coding scheme for each of the 4M partitions in M-QAM constellation space is defined by an 8-bit number as $s_y y_2 y_1 y_0\_\ s_x x_2 x_1 x_0$, where s_x and s_y are the sign bits and are assigned based on the location of partition with respect to the four quadrants of Λ_R space. We define the real and imaginary 4-bit partition codes as:

$$p_r = s_x x_2 x_1 x_0 \qquad (4\text{-}27)$$
$$p_i = s_y y_2 y_1 y_0$$

It should be noted that the constellation space Λ (Λ_R) is also symmetrical with respect to the real and imaginary axes. Hence the problem of finding the closest constellation point to C_{i+1} can be solved in the first quadrant. In order to address the original problem of finding the eight neighbouring constellation points and ordering them; all four quadrants of Λ_R space have to be considered [64].

Enumeration Algorithm

Step 1

Find location of C_{i+1} in Λ_R space by identifying real and imaginary partition codes for C_{i+1}. As mentioned before, in order to reduce the complexity, we first locate $|C_{i+1}|$ in the first quadrant. This will reduce the required number of comparators to 14 (7 for each of the real and imaginary axis).

Step 2

From the real and imaginary partition codes, find centre point $\mathbf{u_0}$ as the closest constellation point to C_{i+1}. As it can be seen in Figure 4.25, there are four neighbouring partitions that point to $\mathbf{u_0}$ as their closest point. The code for constellation point is defined as $s_y y_{c1} y_{c0}\_ s_x x_{c1} x_{c0}$, where s_x and s_y are the sign bits in partition codes. The constellation code for $\mathbf{u_0}$ is derived from the partition code as:

$$\mathbf{u_{0r}} = x_{c1}x_{c0} = x_2x_1 \qquad \textbf{(4-28)}$$
$$\mathbf{u_{0i}} = y_{c1}y_{c0} = y_2y_1$$

Having determined the closest constellation point $\mathbf{u_0}$ in the first quadrant, the sign of real and imaginary parts of C_{i+1} can be used to reverse the mapping to the first quadrant.

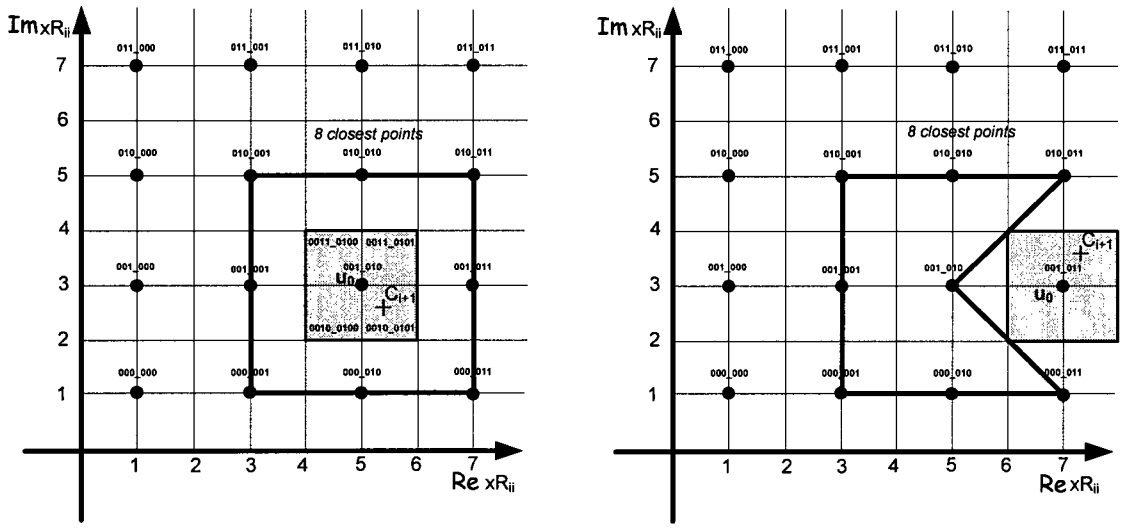


Figure 4.25 Neighbouring constellation points

Step 3

Find the eight neighbouring constellation points. In order to find neighbouring constellation points, the real and imaginary constellation codes for centre point $\mathbf{u_0}$ have to

be incremented and decremented by one. This process should include the boundary cases where the centre point $\mathbf{u_0}$ is located on the boundaries of the constellation space Λ_R. Figure 4.25(b) shows an example of a corner case where the centre point location prohibits simple increment, in this figure eight neighbouring points are depicted. The modified definition for increment and decrement should include handling of corner cases. Table 4.3 defines the modified increment and decrement operations for 64-QAM constellation space. The boundary cases are 011 and 111 entries. The result of incrementing 011 is 001 and decrementing 111 results in 101. As an example in Figure 4.25(b), the code for $\mathbf{u_0}$ is 001_011; incrementing 011 points to location 001_001.

Table 4.3 Modified increment and decrement for 64-QAM constellation

$s_x x_{c1} x_{c0}$ or $s_y y_{c1} y_{c0}$	Increment operation (++)	Decrement Operation (--)
111	110	101
110	101	111
101	100	110
100	000	101
000	001	100
001	010	000
010	011	001
011	001	010

Once the centre and eight neighbouring points are identified, the neighbouring constellation points are ordered based on their respective metric values. The neighbouring points that do not satisfy the sphere radius test are eliminated and finally PEDs for all the selected child nodes are calculated.

Position of C_{i+1} with respect to the geometric line $x=y$ defines two decision boundaries that are used to enumerate the eight neighbouring constellation points. In Figure 4.26; x and y are defined as:

$$x = \left\| \Re\{C_{i+1}\} \right| - \left| \Re\{\mathbf{u}_0\} \right\| \qquad \textbf{(4-29)}$$
$$y = \left\| \Im\{C_{i+1}\} \right| - \left| \Im\{\mathbf{u}_0\} \right\|$$

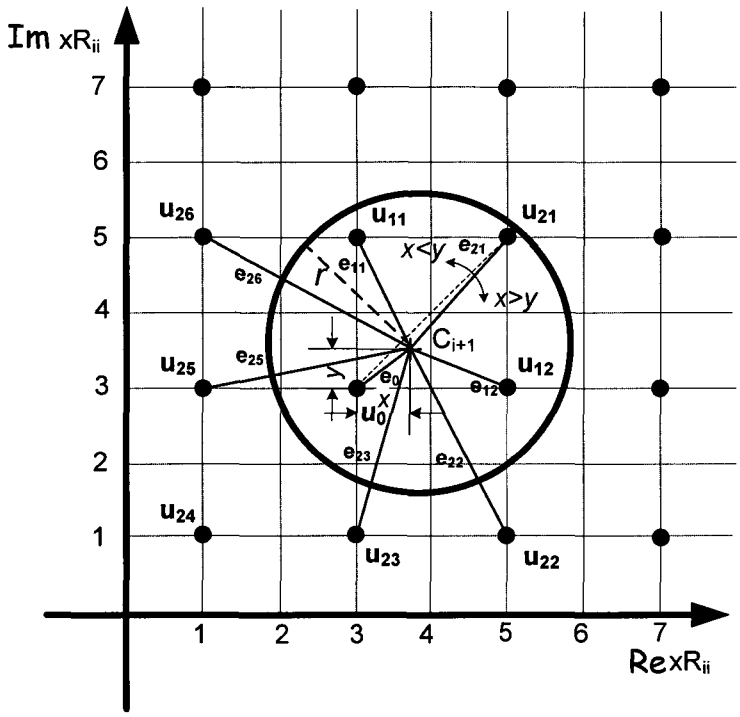


Figure 4.26 Metric enumeration of closets 9 constellations

The distances between the nine neighbouring constellation points and C_{i+1} are depicted in Figure 4.26 and can be derived using l^{∞}-norm definition in equation (4-24) as follows:

$$
\begin{aligned}
\mathbf{e}_0 &= \max(x, y) \qquad \textbf{(4-30)}\\
\mathbf{e}_{11} &= \max(x, (2-y)) = 2-y\\
\mathbf{e}_{12} &= \max((2-x), y) = 2-x\\
\mathbf{e}_{21} &= \max((2-x),(2-y)) = \begin{cases} 2-x, & x \le y \\ 2-y, & x > y \end{cases}\\
\mathbf{e}_{22} &= \max((2-x),(2+y)) = 2+y\\
\mathbf{e}_{23} &= \max(x,(2+y)) = 2+y\\
\mathbf{e}_{24} &= \max((2+x),(2+y)) = \begin{cases} 2+y, & x \le y \\ 2+x, & x > y \end{cases}\\
\mathbf{e}_{25} &= \max((2+x), y) = 2+x\\
\mathbf{e}_{26} &= \max((2+x),(2-y)) = 2+x
\end{aligned}
$$

Based on the two decision boundaries ($x<y$) or ($x>y$) the order of metric increments can be stated as:

$$
\begin{cases} x \le y, & \{\mathbf{e}_0, \mathbf{e}_{11}, \mathbf{e}_{12}, \mathbf{e}_{21}, \mathbf{e}_{25}, \mathbf{e}_{26}, \mathbf{e}_{23}, \mathbf{e}_{22}, \mathbf{e}_{24}\} \\ x > y, & \{\mathbf{e}_0, \mathbf{e}_{12}, \mathbf{e}_{11}, \mathbf{e}_{21}, \mathbf{e}_{23}, \mathbf{e}_{22}, \mathbf{e}_{25}, \mathbf{e}_{26}, \mathbf{e}_{24}\} \end{cases} \qquad \textbf{(4-31)}
$$

Consequently SE ordering for the constellation points is as stated in (4-32).

$$\begin{cases} x \le y, & \{\mathbf{u}_0, \mathbf{u}_{11}, \mathbf{u}_{12}, \mathbf{u}_{21}, \mathbf{u}_{25}, \mathbf{u}_{26}, \mathbf{u}_{23}, \mathbf{u}_{22}, \mathbf{u}_{24}\} \\ x > y, & \{\mathbf{u}_0, \mathbf{u}_{12}, \mathbf{u}_{11}, \mathbf{u}_{21}, \mathbf{u}_{23}, \mathbf{u}_{22}, \mathbf{u}_{25}, \mathbf{u}_{26}, \mathbf{u}_{24}\} \end{cases} \quad (4\text{-}32)$$

Based on sphere radius test updated metric T_i should satisfy the conditions in (4-33).

$$T_i = \max(T_{i+1}, e_i) \le r \quad (4\text{-}33)$$

$$T_i = \begin{cases} I : T_{i+1} \le r, & e_i \le T_{i+1} \\ II : e_i \le r, & e_i > T_{i+1} \end{cases}$$

Condition *I* in (4-33) is always true since T_{i+1} is the metric value from the $(i+1)$-th tree level which has already passed sphere radius test and only condition *II* has to be evaluated. In (4-30) all possible values of e_i have been derived. In order to test condition *II* in (4-33) the inequalities in (4-34) have to be evaluated.

$$\begin{cases} I_x : x \le r & I_y : y \le r \\ II_x : x \ge 2 - r & II_y : y \ge 2 - r \\ III_x : x \le r - 2 & III_y : y \le r - 2 \end{cases} \quad (4\text{-}34)$$

Each of the inequalities in (4-34) can be considered as a test case for passing sphere radius test. In Table 4.4 these test cases have been listed along with the two decision boundaries for $x<y$ and $x>y$. If a test case is valid; the metrics listed under the decision boundary column pass the sphere radius test.

Table 4.4 Sphere radius test

Test Case	*Decision Boundary*	
	$x \le y$	$x > y$
I_x	-	e_0
II_x	e_{12}, e_{21}	e_{12}
III_x	e_{25}, e_{26}	e_{25}, e_{26}, e_{24}
I_y	e_0	-
II_y	e_{11}	e_{11}, e_{21}
III_y	e_{23}, e_{22}, e_{24}	e_{23}, e_{22}

4.8.2 Hardware architecture

In this section a low complexity parallel architecture for *LCCE* enumeration algorithm will be introduced. The top-level block diagram of *MECU* block has been depicted in Figure 4.27. *MECU* architecture has been broken into four major blocks; *mapping*, *enumeration*, *sphere constraint* and *metric computation*. The data input to the block diagram is C_{i+1} which can be expressed based on the mapped received signal y_{qi} as defined in equation (4-5) and upper triangular channel matrix **R** as

$$C_{i+1} = y_{qi} - \sum_{j=i+1}^{N_t} R_{ij} s_i \qquad (4\text{-}35)$$

In *mapping* block the received signals are mapped into M-QAM constellation space and the neighbouring constellation points are determined.

Enumeration block orders the neighbouring constellation points based on the method presented in step 3 of the enumeration algorithm in section 4.8.1. *Sphere constraint* and *metric computation* blocks use the center point ($\mathbf{u_0}$) information provided by the *enumeration* block to asses the validity of each of the neighbouring points based on sphere radius test and also compute the updated metric values ($T_i(\mathbf{u}_j)$). Valid bits $\mathbf{vu_j}$ show the result of sphere radius test for each of the selected constellation points ($\mathbf{u_j}$) and will be saved in memory along with updated metrics and partial symbol vectors.

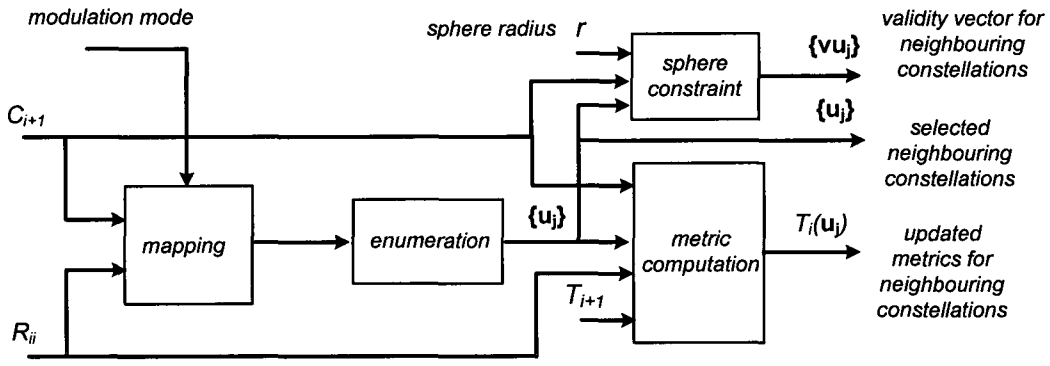


Figure 4.27 Top-level block diagram of MECU block

Figure 4.28 shows C_{i+1} computation circuit which operates based on equation (4-35). Different rows in upper triangular matrix **R** are selected by *level* signal. Each row is multiplied by elements of partial symbol vector, i.e. $\mathbf{s}_4$, $\mathbf{s}_3$, $\mathbf{s}_2$. In order to reduce circuit complexity, complex multipliers are implemented using only add-and-shift operations similar to the architecture in Figure 4.10.

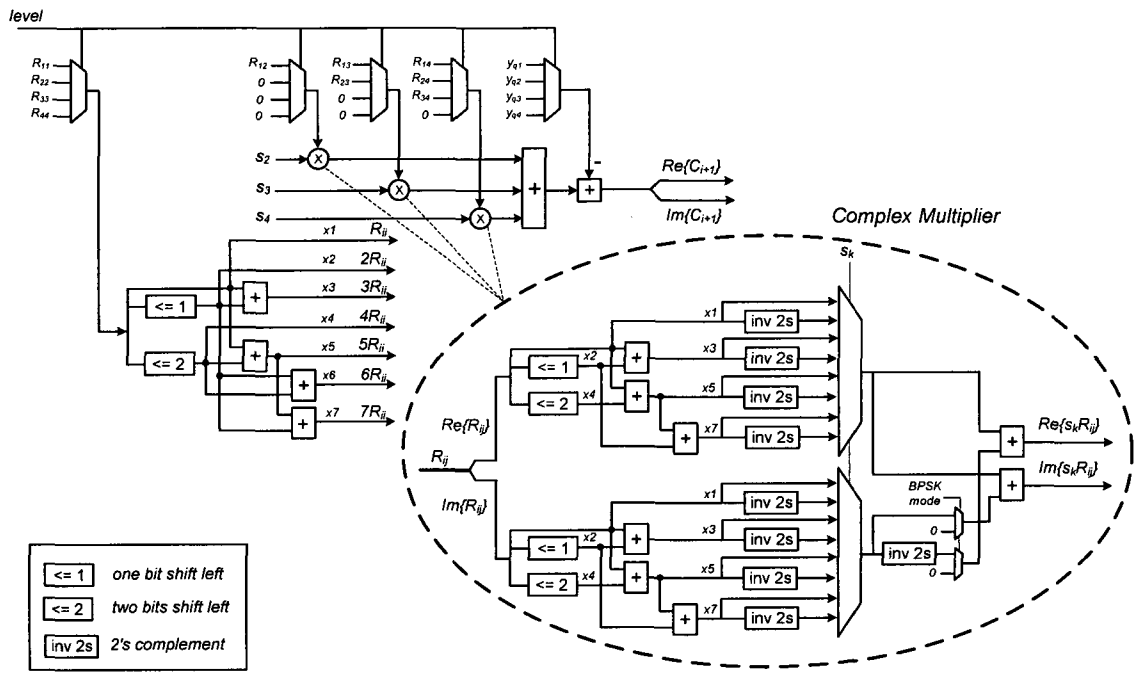


Figure 4.28 Center point C_{i+1} computation

Mapping- Enumeration

Mapping and *enumeration* blocks are depicted in Figure 4.29. Two sets of parallel decoders compare $|Re\{C_{i+1}\}|$ and $|Im\{C_{i+1}\}|$ with weighted R_{ii} values to determine the location of $|C_{i+1}|$ in Λ_R space. The results of decoding are the 3-bit partition codes x_{sq} and y_{sq} which identify the partition in which $|C_{i+1}|$ is located as depicted in Figure 4.25. The partition codes are later used in *constellation mapping* block to find the closest constellation point to C_{i+1} or centre point $\mathbf{u_0}$.

The *increment* and *decrement* blocks are combinational logic blocks and their logic functions have been defined in Table 4.3. The real and imaginary components of $\mathbf{u_0}$ ($\mathbf{u_{0r}}$, $\mathbf{u_{0i}}$) are incremented and decremented in order to find all the neighbouring constellation points surrounding centre point $\mathbf{u_0}$. The closest constellations to C_{i+1} have been indentified as $\{\mathbf{u_0},\mathbf{u_{11}},\mathbf{u_{12}},\mathbf{u_{21}},\mathbf{u_{22}},\mathbf{u_{23}},\mathbf{u_{24}},\mathbf{u_{25}},\mathbf{u_{26}}\}$ in Figure 4.29.

Position of C_{i+1} with respect to geometric line $x=y$ defines two decision boundaries which are used to enumerate eight neighbouring constellation points. SE ordering for the constellation points is stated in (4-36).

$$\begin{cases} x \le y, & \{\mathbf{u_i}\}_{i=0}^{8} = \{\mathbf{u_0},\mathbf{u_{11}},\mathbf{u_{12}},\mathbf{u_{21}},\mathbf{u_{25}},\mathbf{u_{26}},\mathbf{u_{23}},\mathbf{u_{22}},\mathbf{u_{24}}\} \\ x > y, & \{\mathbf{u_i}\}_{i=0}^{8} = \{\mathbf{u_0},\mathbf{u_{12}},\mathbf{u_{11}},\mathbf{u_{21}},\mathbf{u_{23}},\mathbf{u_{22}},\mathbf{u_{25}},\mathbf{u_{26}},\mathbf{u_{24}}\} \end{cases} \quad \textbf{(4-36)}$$

The multiplexers in *reordering* block (Figure 4.29) select between the two cases in equation (4-36) and complete the enumeration of child nodes.

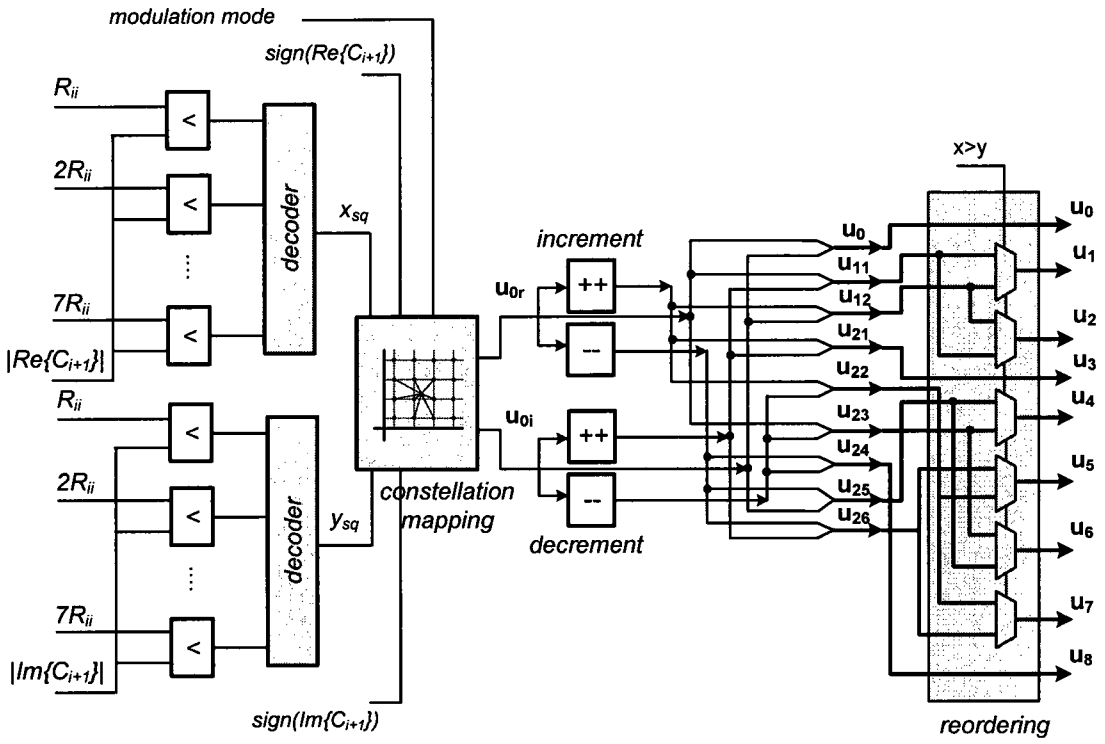


Figure 4.29 Decoding-mapping and enumeration blocks

Constellation mapping

Configurable *constellation mapping* block is shown in Figure 4.30. This block uses *partition codes* (x_{sq}, y_{sq}) and *modulation mode* to determine the closest constellation point to C_{i+1} (centre point $\mathbf{u_0}$). *Modulation mode* signal determines type of modulation scheme for received symbols.

In step 2 of the enumeration algorithm in section 4.8.1 the process of mapping has been stated as

$$\mathbf{u}_{0r}[1:0] = x_{c1}x_{c0} = x_{sq}[2:1] \qquad \text{(4-37)}$$
$$\mathbf{u}_{0r}[2] = sign(\mathrm{Re}\{C_{i+1}\})$$
$$\mathbf{u}_{0i}[1:0] = y_{c1}y_{c0} = y_{sq}[2:1]$$
$$\mathbf{u}_{0i}[2] = sign(\mathrm{Im}\{C_{i+1}\})$$

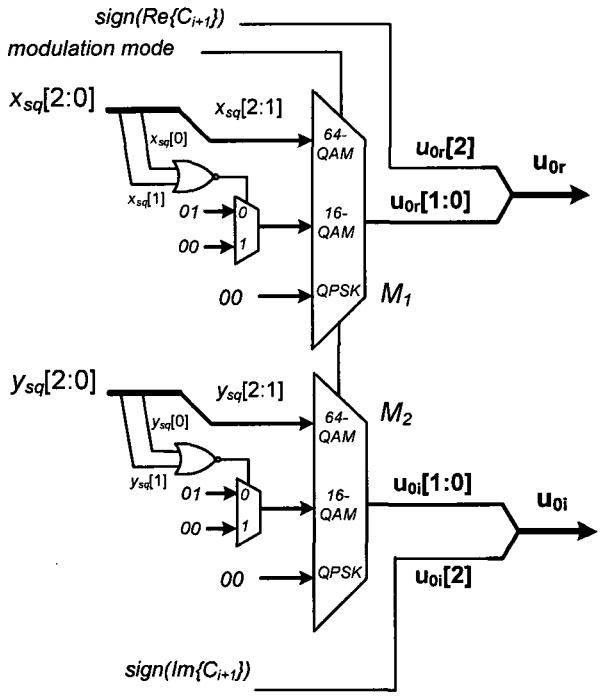


Figure 4.30 Configurable constellation mapping circuit

In case of 16-QAM and QPSK modulation, the mapping space is a subset of 64-QAM space. The process of limiting the mapping space can be well described using a graphical example. In Figure 4.31 two different examples of mapping C_{i+1} to centre point $\mathbf{u_0}$ for 64-QAM, 16-QAM and QPSK modulation modes have been demonstrated. In both cases C^A_{i+1} and C^B_{i+1} are located within the constellation space for 64-QAM modulation and outside the space for 16-QAM and QPSK modulation space. The mappings in both cases have shown for different modulation schemes using the arrows. In case of 16-QAM modulation only $x_{sq}[1:0]$, $y_{sq}[1:0]$ are used to determine mapping to four constellation points located in the first quadrant of constellation space. QPSK modulation scheme has only one constellation point located in the first quadrant and the result of mapping is one single point. *M1* and *M2* multiplexers in Figure 4.30 limit the mapping space for different types of modulation schemes.

BPSK modulation is considered as special case of QPSK modulation in Figure 4.31 since the real parts of BPSK constellations is either (1) or (-1) and the imaginary part is zero. In Figure 4.30, BPSK modulation mode for real part is handled the same way as QPSK

modulation. In Figure 4.28, the *BPSK mode* signal in complex multipliers set the imaginary part of the products to zero if s_2, s_3 or s_4 are BPSK signals.

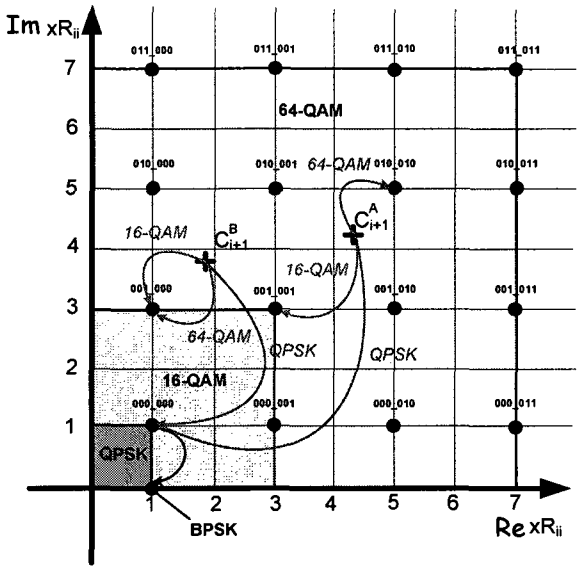


Figure 4.31 Constellation mapping examples for different modulation schemes

Metric computation

In Figure 4.32 architecture of *metric computation* block is shown. This block computes the metric increments of each of the neighbouring constellation points as well as the updated PED values. This architecture has been designed based on step 3 of the enumeration algorithm in section 4.8.1 and equation (4-30).

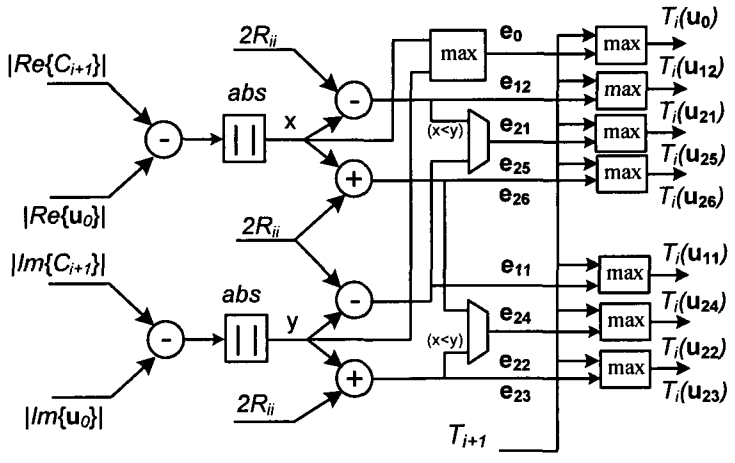


Figure 4.32 Metric computation

Sphere constraint

Step 3 of the enumeration algorithm also includes a procedure for sphere radius test in order to enumerate only those constellation points that are located within the test sphere. The circuit for sphere radius test is shown in Figure 4.33. The validity of each neighbouring constellation point is characterized by a status bit $\mathbf{vu_{ij}}$ as shown in Figure 4.33 and defined in Table 4.4. The status bits are rearranged in *reordering* block as stated in equation (4-36) and shown in Figure 4.29. In case of QPSK and BPSK modulation modes, the status bits for the unused neighbouring constellations are set to zero.

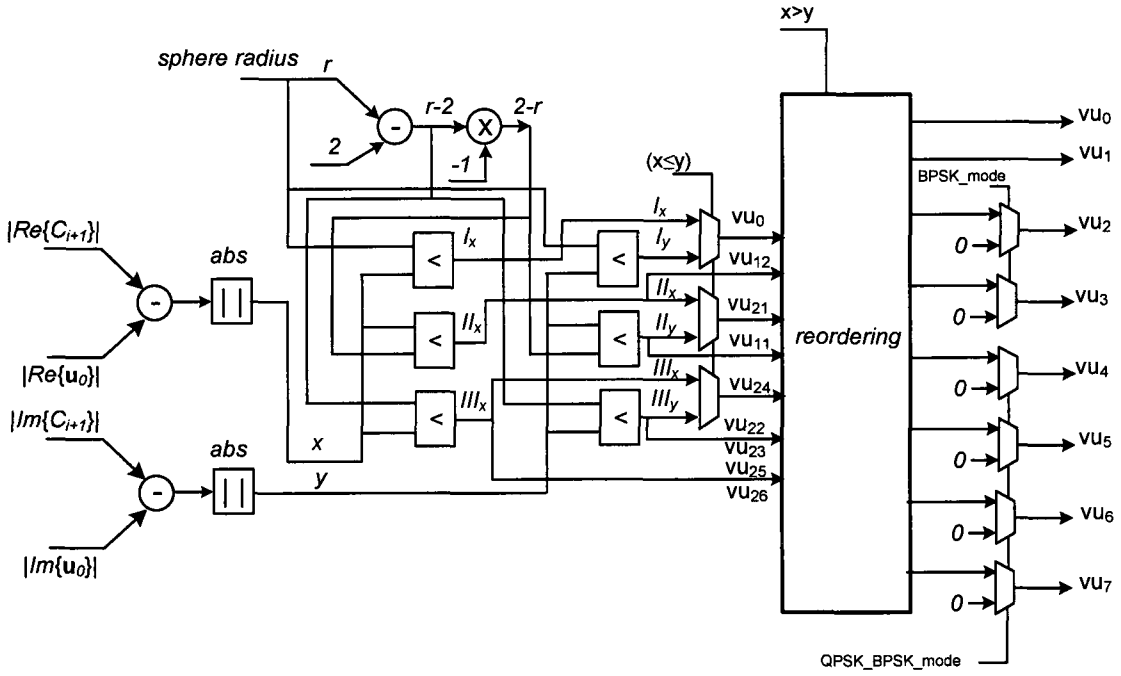


Figure 4.33 Sphere radius test

The MECU architecture has been synthesized for different delay constraints using 0.13μm CMOS standard cell library and the resulting delay-area curve has been depicted in Figure 4.34. The area figures are based on equivalent gate numbers. The horizontal axes represent the circuit delay constraint which is inversely proportional to the maximum operating clock frequency. The maximum achieved operating frequency of MECU block in 0.13μm CMOS is 160 MHz.

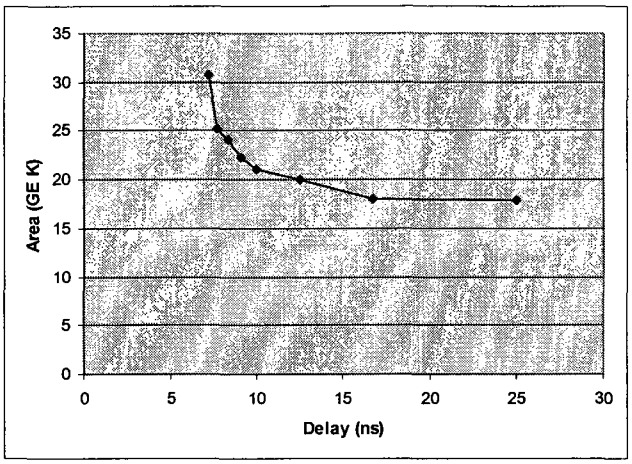


Figure 4.34 Delay-Area curve for MECU architecture

4.9 Dynamically configurable MECU based detector architecture

In section 4.5.2 a configurable depth-first MIMO detector (DFSD architecture) based on exhaustive enumeration has been introduced and later in sections 4.7 and 4.8 low complexity DFKB and LCCE algorithms have been presented as solutions to improve the throughput of depth-first sphere decoding algorithm for higher order modulation schemes. In this section a novel MIMO detector architecture based on *MECU* processing element and DFKB(8-4-4) algorithm will be introduced. In DFKB(8-4-4) algorithm, eight child nodes on level 4 and four child nodes on levels 3 and 2 are selected and saved for further processing. Sphere radius in this implementation varies based on tree level. The Level-based sphere radius (LBSR) has been discussed in section 4.7.3.

The top-level detector architecture of DFKB-MECU has been shown in Figure 4.35. This architecture consists of four blocks that are arranged in type II folded architecture. In type II folded architecture as depicted in Figure 4.4; there is only one metric enumeration block which is located after the metric computation block. All the child nodes are first enumerated and their associated metrics are sorted before being stored in the memory. This simplifies architecture of memory controller since there is no need for additional enumeration of child nodes in memory which was the case in type I architecture as implemented in DFSD MIMO detector in section 4.5.2.

Metric Computation and Enumeration Unit (MECU) is the computation engine of MIMO detector. Architecture of *MECU* block has been discussed in section 4.8.2. The incoming

received symbols after preprocessing (y_{qi}) (as defined by equation (4-5)) enter *MECU* block. Input partial symbol vector $\mathbf{P}_{i+1}$ is the list of all selected nodes from upper tree levels and T_{i+1} is accumulated metric (PED) of the selected parent node. Other inputs to *MECU* block are channel parameters ($\mathbf{R}_{ij}$) and *modulation mode* signal which represent modulation scheme for each of the transmitted symbols. The outputs of *MECU* block are eight child nodes of the input parent node with the lowest accumulated metrics $T_i(\boldsymbol{u}_m)$ *(i=4,3,2,1 and m=0,1,...7)* arranged in the order of magnitude which is reflected in their partial symbol vectors $P_i(\boldsymbol{u}_m)$ *(i=4,3,2,1 and m=0,1,...7)*. The valid bits for selected child nodes that pass sphere radius test have been represented by $V_r(\boldsymbol{u}_m)$ and are saved in *Path History Buffer (PHB)* block along with the metrics and partial symbol vectors.

Control Unit (*CU*) block is the control centre of the architecture and controls start, stop and data flow through the architecture. It also determines forward and backward movements of the search along the search tree by controlling multiplexer *M1* and *M2*. If the data from *PHB* block is fed into *MECU* block, the search back tracks to a parent node on an upper level or process a new parent node from the same level of the tree. If the output of the *MECU* block is directly fed back into it after one clock cycle delay, the search moves forward along the tree graph to a lower tree level.

The search is over when there are no more saved nodes left in *PHB* and *empty* signal is asserted. Detected symbols are saved in *Symbol Buffer* and detector is ready to process a new set of symbols.

Partial symbol vectors and their associated metrics (T_i) are stored in *Path History Buffer* (*PHB*) block. Multiple data vectors can be written to *PHB* simultaneously. *Push* and *pop* signals from *CU* controls write and read operations.

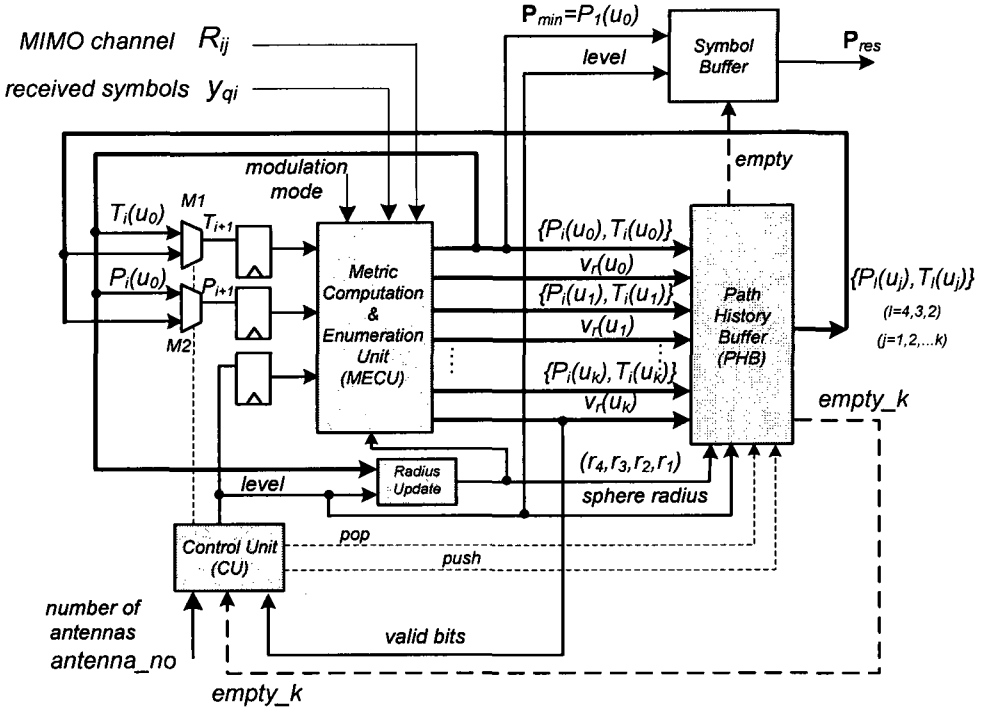


Figure 4.35 Top-level architecture DFKB-MECU detector

Architecture of Path History Buffer (PHB) shown in Figure 4.36 has been simplified compared to the architecture of *Data Buffer* shown in Figure 4.13 and Figure 4.14. *Data Buffer* block in the reference architecture for depth-first sphere decoder presented in section 4.5.2 performs three main tasks:

- Saving intermediate search data
- Selection of the next parent node from the list of saved nodes (enumeration)
- Sphere radius test

In *MECU* based architecture; metrics have already been sorted by *MECU* block and as a result there is no need for the second level of enumeration in *PHB* block as well as searching saved nodes in each memory module for the smallest metric value. An address *decoder* in each buffer module sequentially addresses valid buffer locations.

The internal filtering mechanism still exists and continuously searches for all the metric values that violate sphere radius test and removes them from the buffer. The valid bit for a buffer location is set to '0' once the sphere radius is violated by the associated metric

value or the buffer location is being read. When all the valid bits in a buffer module are set to '0', *empty_k* signal is asserted showing that the buffer module is empty.

In DFKB decoding algorithm as discussed in section 4.7.2 only limited number of nodes are saved and the rest are discarded. In Figure 4.36 has been implemented for a DFKB (8-4-4) based algorithm. This means that at the fourth level; the metric results for eight selected nodes are saved and at level 3 and level 2 the results for four selected nodes are stored in the buffer. As a result in Figure 4.36, the BUF_4 module has eight addressable cells while modules BUF_3 and BUF_2 each have four addressable cells. The total number of addressable memory locations in this architecture for 64-QAM implementation is 16 which shows 60% reduction in memory space compared to *Data Buffer* block shown in Figure 4.13 which has been implemented for 16-QAM modulation.

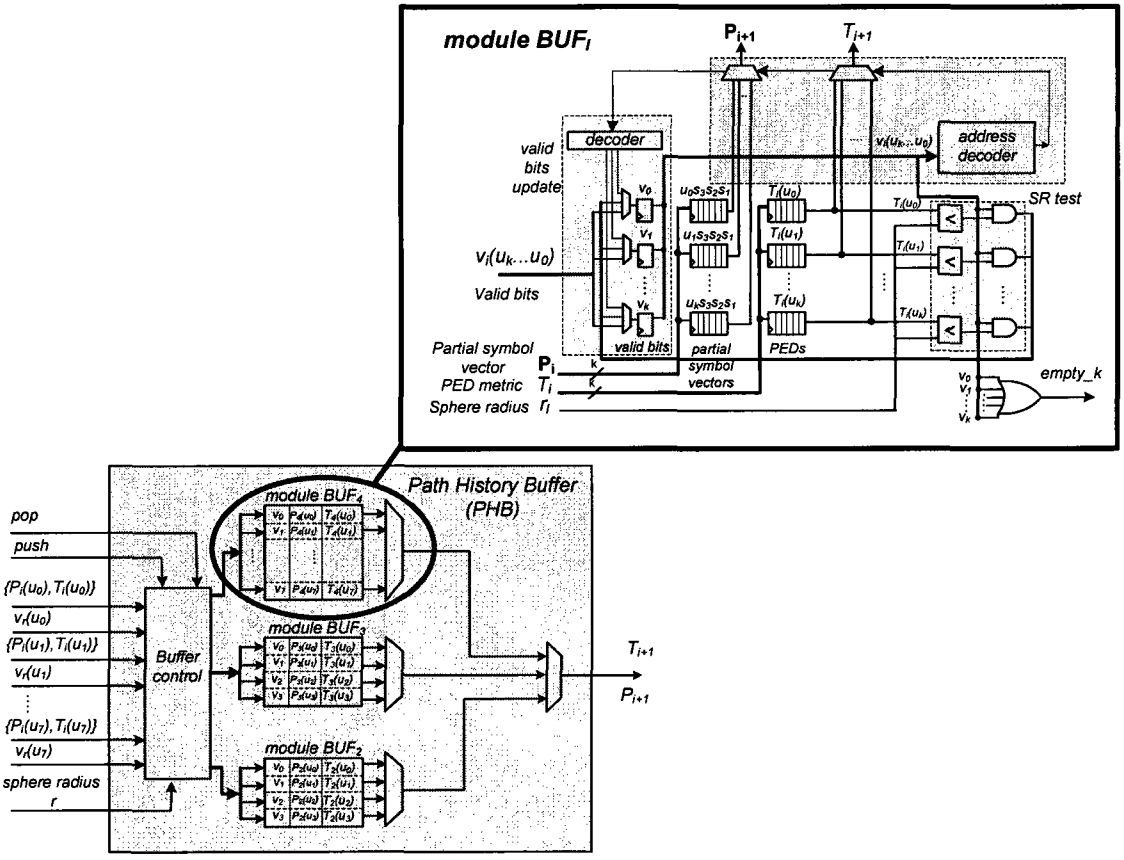


Figure 4.36 PHB buffer

Control Unit is similar to the control block in DFSD detector in section 4.5.2. The only difference is that in DFKB based architecture, the modulation mode reconfiguration is in *MECU* block. *Buffer Control* state machine as depicted in Figure 4.15 remains

unchanged. It controls forward and backward movements of the search as well as *push* and *pop* operations.

Radius Update block updates the sphere radius when the search reaches the first level of the search tree. If the number of iterations is more than five, the level based sphere radius (LBSR) definition as described in section 4.7.3 is being used for sphere radius updates.

In Figure 4.37 the average processing delay of the DFKB(8-4-4) based architecture has been plotted and compared to a depth first sphere detector in 4x4 64-QAM operating mode. The average throughput of the detector is inversely proportional to the symbol processing delay as stated by

$$DR = \frac{N_t Q_b f_{clk}}{D_{avg}} \tag{4-38}$$

Where D_{avg} is the average symbol processing delay in cycles, N_t is the number of transmitting antennas, Q_b is the number of bits per symbol and f_{clk} is the operating clock frequency of circuit. The processing delay in DFKB-MECU(8-4-4) architecture has been reduced by up to 68% at lower signal to noise ratios compared to depth-first sphere detector.

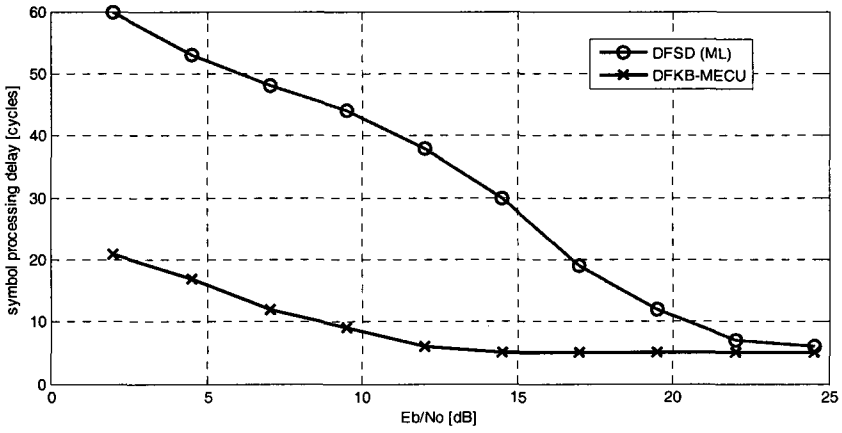


Figure 4.37 Symbol processing delay in DFKB-MECU and DFSD (ML) detectors

The BER performance results are presented in Figure 4.38. In this figure the BER performance of DFKB-MECU(8-4-4) architecture has been compared with ML detector performance and also depth-first sphere decoder (DFSD) with l^{∞}-norm definition. The BER performance loss in DFKB-MECU architecture is 1.1 dB compared to ML detector and less than 0.1 dB compared to DFSD algorithm with l^{∞}-norm definition.

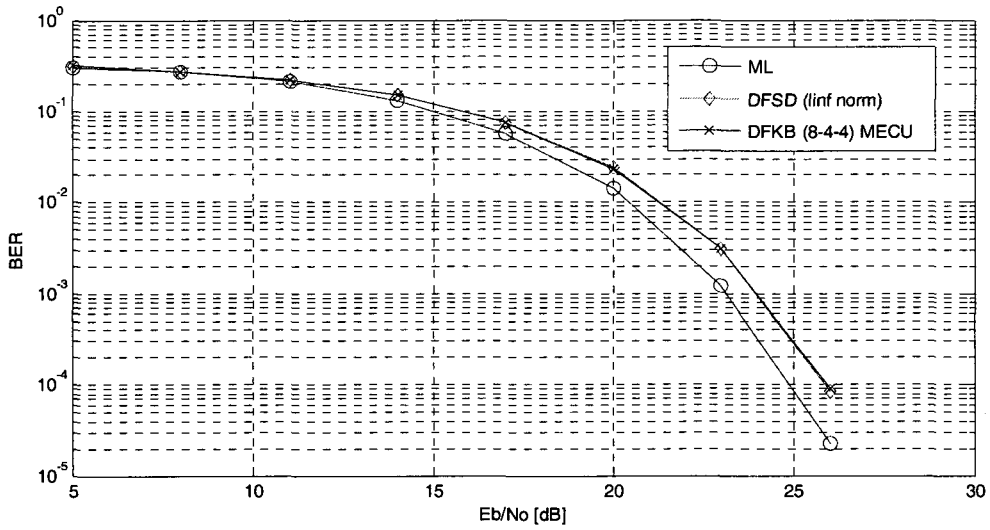


Figure 4.38 BER performance of DFKB-MECU receiver

In section 4.12 the physical implementation results for DFKB-MECU architecture are presented and discussed.

4.10 Array processing architecture for fixed sphere decoding

The main idea in Fixed Sphere Decoding algorithm (FSD) is to reduce the computational complexity of ML detector by limiting number of search nodes to a subset of the constellation space Λ. Fixed sphere decoding follows the search independent of the noise level over a fixed number of lattice points **Hs** located within a subset $S \subset \Lambda$. In this algorithm a fixed number of child nodes n_i from each of the parent nodes are selected for further processing. The total number of candidate symbols to be processed is $N_s = \prod_{i=1}^{N_t} n_i$

The BER performance of this algorithm without channel reordering is medium to low (Table 3.1). The n_i candidates on each level of the search tree are selected according to increasing distance to y_{qi} based on SE enumeration principle.

Figure 4.39 shows the tree search diagram of a FSD (K,1,1,1) ($K \leq 8$) in a 4x4 system with 64-QAM modulation. On level four of search tree nodes $\mathbf{s}_0$~$\mathbf{s}_7$ are the selected child nodes with the smallest metrics. These nodes will be expanded to their child nodes on level 3. The child node with the smallest metric is selected and the search continues. On

level one of the search tree, the metrics of the final eight nodes are compared to find the search result.

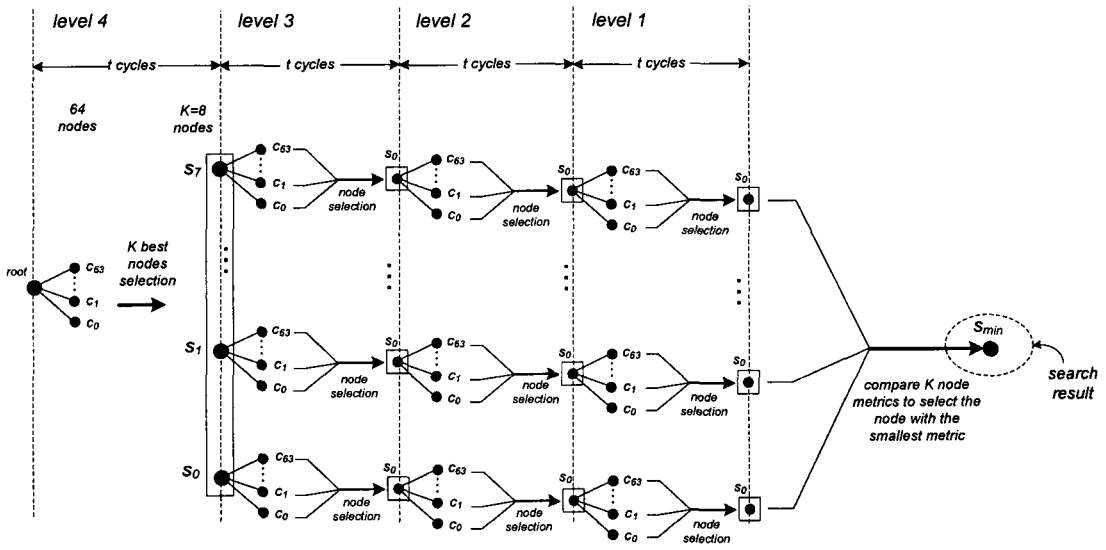


Figure 4.39 Tree search diagram- FSD (K,1,1,1) in 4x4 64-QAM system

Fixed sphere decoding (FSD) is inherently a parallel algorithm and can be best implemented using an array of processing elements. Each of the parallel streams from level 3 to level 1 in the tree search diagram in Figure 4.39 can be considered as a cascade of processing elements that operate in parallel. There is a trade-off between the number of selected candidates at each level and performance of FSD algorithm, such that by increasing the number of selected candidates the BER performance of decoder improves.

The main architectural challenge in implementation of FSD algorithm is finding n_i candidates with the smallest metrics. Metric enumeration and computation block (*MECU*) introduced in section 4.8.2 can be used as the main processing unit of an array processing architecture for implementation of FSD algorithm with low complexity. Due to the architectural design of MECU block, implementation of different variants of FSD algorithm for FSD (K_4,K_3,K_2,K_1) such that $K_i \leq 9$ is also possible. In this section first a VLSI architecture for FSD (K,1,1,1) detector will be introduced and later this architecture will be expanded to the more general (K_4,K_3,K_2,K_1) case.

FSD (K,1,1,1) detector depicted in Figure 4.40 consists of K processing elements (PE) working in parallel. Each processing element is responsible for processing child nodes of one of the K selected parent nodes at level 4 of search tree. The K selected metrics from

each of K processing elements are compared in *min_k* block to determine the minimum metric and detected transmitted symbols.

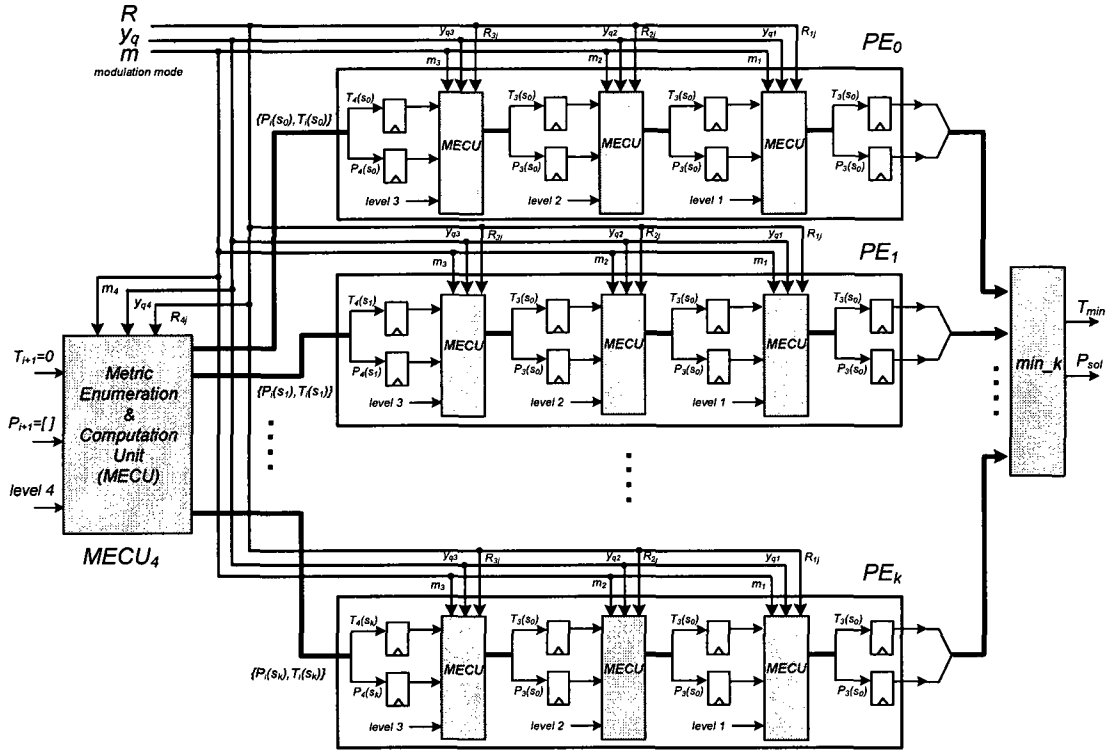


Figure 4.40 FSD (*K*,1,1,1) detector architecture

Each processing element consists of three *MECU* blocks separated by buffer stages and is responsible for processing nodes on one level of the search tree. *MECU* blocks can be configured to operate in BPSK, QPSK, 16-QAM or 64-QAM modulation modes. All the nodes on level four of the search tree are processed using only one MECU block that selects the K best metrics. Each of K processing elements is connected to one of the outputs of $MECU_4$ block. Throughput (DR) of FSD detector is a function of operating clock frequency f_{clk}, N_t number of transmitting antennas and Q_b number of bits per symbol and can be expressed as

$$DR = f_{\max} \cdot Q_b \cdot N_t \qquad (16)$$

Latency of the architecture depends on the number of buffer stages in each PE unit and is proportional to the number of receiving antennas N_r.

The array processor architecture presented in Figure 4.40 can be expanded into a general FSD (K_4-K_3-K_2-K_1) case as shown in Figure 4.41. The total number of required *MECU*

blocks in this architecture is $K_4 \cdot K_3 \cdot K_2 \cdot K_1$. The expansion of architecture into the general case does not influence overall data rate and latency of the decoder and it only improves the BER performance of detector.

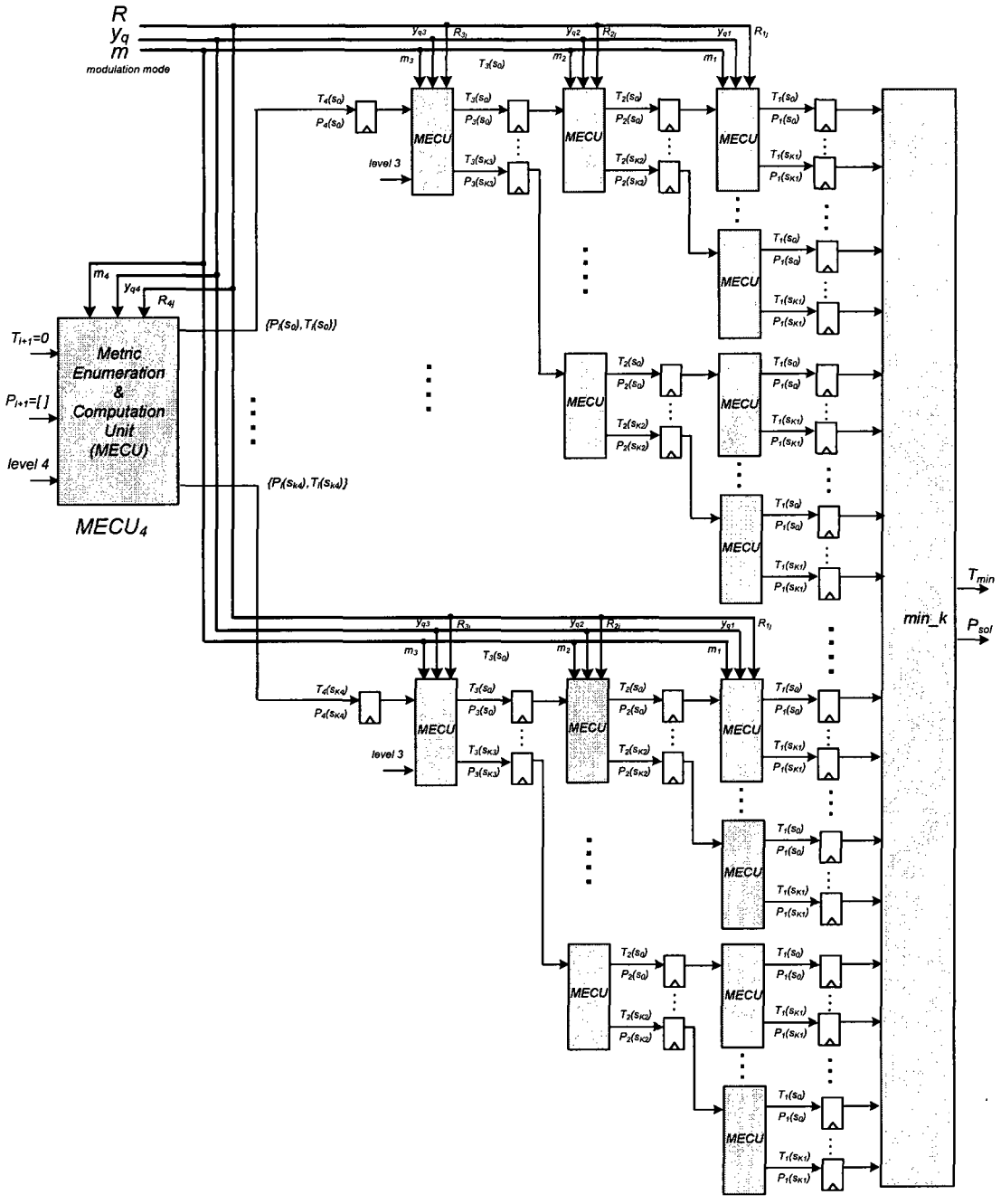


Figure 4.41 FSD (K_4,K_3,K_2,K_1) detector architecture

4.11 Parallel depth-first sphere decoding (PDFKB)

Depth-first sphere decoding is a recursive algorithm in which the metric constraint (sphere radius) dynamically updates during the search. Such dynamic behaviour is a barrier to parallel path searches since parallel paths need to consider the results obtained from the previously followed paths. This results in a sequential search where each path has to be followed to its maximum depth (depending on the metric constraint), before a new search can be started.

In section 4.10 a fully parallel architecture based on FSD algorithm and *MECU* block has been introduced. In the current section a new parallel decoder architecture based on the DFKB-MECU architecture of section 4.9 will be presented. The parallel PDFKB architecture shown in Figure 4.42 has two parallel searchers each similar to DFKB-MECU detector. Each of the searches is responsible for complete processing of all the child nodes down to level 1 for one group of parent nodes on level 4. The nodes on level 4 are distributed between the searchers. The primary searcher first processes all the parent nodes on level 4 and stores the results in BUF_4 of *PHB1* (Figure 4.42). From this moment the two searchers work in parallel by processing different nodes on level 4 and expanding them to their child nodes. The *arbiter* block in Figure 4.42 distributes the parent nodes on level 4 between the two searchers. Each of the searchers expands search tree from different points on level 4 and continues the search depth-first until it reaches level 1 and finds a valid path. The search continues by selecting another node on level 4 and expanding the search tree until all the possible paths on tree have been checked.

The radius update process is slightly different than single searcher detector. In the parallel architecture any searcher that reaches the tree leaves on level 1 and finds a valid solution updates the sphere radius for both of the searchers. Such update strategy is more effective in pruning tree branches since both searcher benefits from the most updated radius.

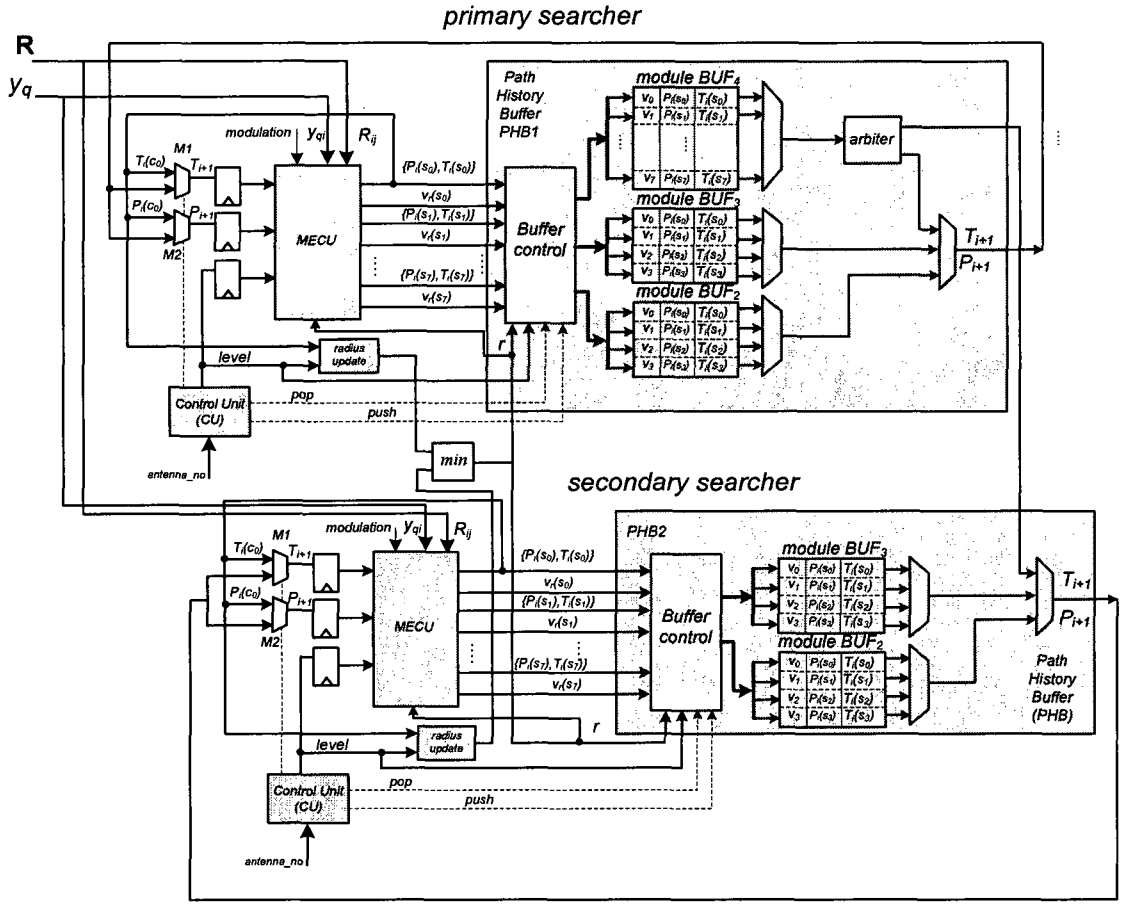


Figure 4.42 Parallel depth-first architecture (PDFKB)

Two node selection strategies can be considered for distribution of parent nodes on level 4 between the two searchers.

Even-Odd distribution: In this strategy, all the even numbered nodes ($\mathbf{s_0}$, $\mathbf{s_2}$, $\mathbf{s_{k-1}}$) are processed by the primary searcher and all the odd numbered nodes ($\mathbf{s_1}$, $\mathbf{s_3}$, $\mathbf{s_k}$) are processed by the secondary searcher. The main benefit of this distribution method is the lower logic complexity of distribution algorithm. The *BUF4* module in *PHB1* can be divided into two parts for storing even and odd numbered nodes. As the read request is asserted from each of the primary and secondary searchers the *arbiter* directs the requests to the two buffers.

In this method request contention is not an issue since the two buffers are operating independently. Request contention happens when two read requests (pop request) from the two searchers are asserted simultaneously. This strategy does not result into an optimum selection of nodes since once any of the two searchers completed processing all

the branches connected to a parent node on level 4, it can not move directly to the next best available parent node. The other benefit of applying even-odd distribution is that it can be easily expanded to higher number of parallel search engines by dividing parent nodes into smaller groups and adding more parallel searchers.

Best-first distribution: In best-first distribution, all the parent nodes on level 4 are stored in one buffer, the *arbiter* block processes the read requests from each of the two search engines and assigns the nodes in the order the requests are received. If two read requests are asserted at the same time, the primary searcher has the priority over the secondary searcher. This may cause one clock cycle delay in secondary searcher operation. The best-first arbitration is the optimum algorithm since it exactly follows SE enumeration strategy, i.e. the nodes with the smallest accumulated metrics is processed first.

Figure 4.43 shows the symbol processing delay of parallel DFKB architecture compared to single DFKB (8-8-8) architecture which performs up to 30% better at lower signal to noise ratios.

In Figure 4.44 the BER performance of FSD and parallel DFKB architectures have been compared. FSD algorithm has higher BER compared to parallel DFKB. It is possible to improve the BER of FSD algorithm by applying channel reordering which adds to the overall complexity of detector [73].

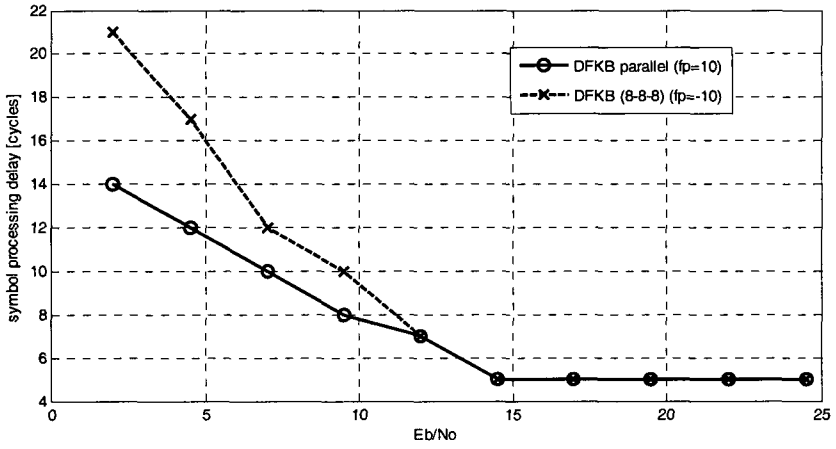


Figure 4.43 Symbol processing delay comparison (Parallel DFKB vs. DFKB)

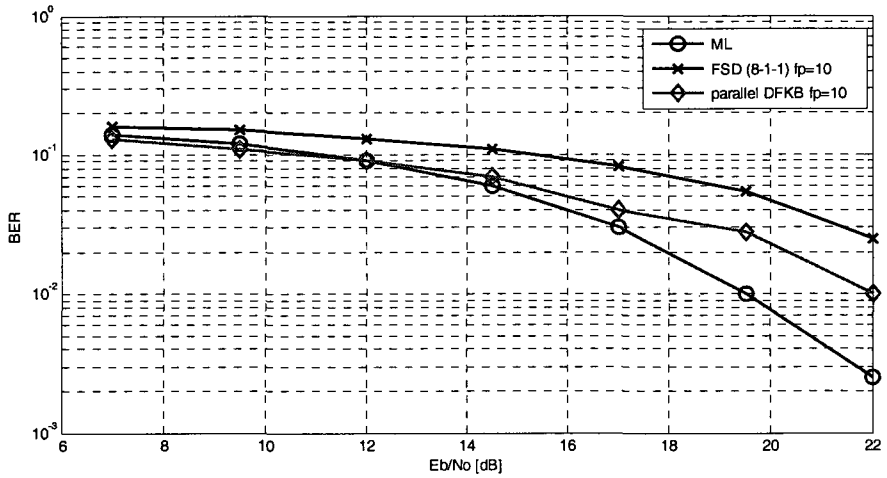


Figure 4.44 BER performance-Parallel DFKB and FSD(8-1-1)

4.12 Implementation results

In Table 4.5; design complexities in equivalent gate numbers, maximum clock frequencies and throughputs for four decoder architectures introduced in this chapter and two configurable architectures reported in the literature have been compared. Equivalent gate numbers are derived by dividing total cell area of the synthesized design by the area of a 2-input drive-1 NAND gate. Each column in Table 4.5 represents the data related to one type of detector defined as follows:

DFSD: Multi-mode configurable depth-first sphere detector with exhaustive search enumeration (section 4.5.2).

DFKB-MECU: Multi-mode configurable detector based on DFKB algorithm with level based radius definition, implemented using *MECU* processing block (section 4.9).

PDFKB: Parallel implementation of DFKB-MECU architecture using two parallel DFKB-MECU designs and even-odd node distribution (section 4.11).

FSD-MECU: Array processing architecture based on FSD (8-1-1-1) parallel algorithm and MECU block as the main processing element (section 4.10).

Huang [70]: Configurable architecture supporting QPSK, 16-QAM and 64-QAM modulation schemes. MIMO detector is composed of a core detector and a residual detector. The core detector is a 2x2 V-BLAST ML detector used for detection of the first two symbols. The residual detector is an ordered successive cancelation (OSIC) detector

that detects the rest of the transmitted symbols. The performance in ASIC-II has been improved by adding a parallel search in V-ML core detector and also by using additional pipelines between functional blocks.

Lin [66]: A reduced complexity sphere detector based on a combination of SD algorithm and successive cancellation detection. In this approach, the complexity of candidate searching has been reduced by shifting the center of constellation space with scalable radius.

Throughput results are presented at 25 dB and 15 dB signal to noise ratio levels and in 16-QAM mode for DFSD architecture and 64-QAM modulation mode for other architectures. Comparing equivalent gate numbers and maximum clock frequency in Table 4.5 shows that DFKB-MECU architecture has the lowest complexity compared to other architectures. DFKB-MECU architecture achieves higher frequency and throughput at lower gate complexity compared to DFSD architecture. These results also show that by applying the algorithm modifications presented in sections 4.7 and 4.8, we were able to support 64-QAM modulation, increase the maximum frequency and at the same time reduce decoder complexity.

It is possible to improve the overall throughput of DFKB-MECU detector by connecting two or more DFKB-MECU decoders in parallel. The area overhead for adding the second search engine in the PDFKB parallel architecture is 18.5%.

FSD-MECU architecture is a fully parallel architecture and has been implemented using pipelined processing elements. The main advantage of this architecture is high throughput of detector which is independent of the signal to noise ratio. The equivalent gate count of FSD-MECU architecture is 197% more than DFKB-MECU architecture and 150% more than PDFKB architecture. The main drawback of FSD-MECU architecture is the low BER performance of detector as depicted in Figure 4.44.

In this chapter we have showed that by applying low complexity enumeration algorithm described in 4.8.1 performance and complexity improves significantly compared to the original DFSD algorithm.

The *Area Efficiency* factor is defined as the ratio of the detector throughput with respect to the equivalent gate count. Area efficiency numbers in Table 4.5 shows more than 3

times improvements in efficiency numbers in *MECU* based architectures compared to *DFSD* architecture.

Table 4.5 Comparison of implementation results

Architecture	This work DFSD	Lin [66]	C. Huang [70]		This work DFKB-MECU	This work PDFKB	This work FSD-MECU
			ASIC I	ASIC II			
Number of Antennas	4×4, 3×3, 2×2	4×4	N_t = 2~4 N_r = 2~6		4×4, 3×3, 2×2	4×4, 3×3, 2×2	4×4, 3×3, 2×2
Modulation	16-QAM, QPSK, BPSK	64-QAM, 16-QAM, QPSK	64-QAM 16-QAM QPSK		64-QAM, 16-QAM, QPSK, BPSK	64-QAM, 16-QAM, QPSK, BPSK	64QAM, 16-QAM, QPSK, BPSK
Detector algorithm	DFSD, (ML)	SD-SIC	OSIC combined V-ML		DFKB LB-radius	DFKB LB-radius	Fixed SD (8-1-1-1)
Tech [μm]	0.13	0.09	0.13		0.13	0.13	0.13
Equivalent gate number	65 K	294 K	190 K	205 K	43 K	51 K	128 K
Maximum Clock Frequency	120 MHz	166 MHz	55 MHz	71 MHz	140 MHz	140 MHz	150 MHz
BER performance[1]	ML	High	Medium	Medium	Sub-optimal	Sub-optimal	High
Throughput @ 15 dB SNR	96 Mbps (4x4 16-QAM)	83 Mbps @ 20 dB (4x4 64-QAM)	36 Mbps	114 Mbps	271 Mbps (4x4 64-QAM)	315 Mbps (4x4 64-QAM)	3600 Mbps (4x4 64-QAM)
Throughput @ 25 dB SNR	264 Mbps (4x4 16-QAM)	95 Mbps (4x4 64-QAM)	36 Mbps	114 Mbps	672 Mbps (4x4 64-QAM)	672 Mbps (4x4 64-QAM)	3600 Mbps (4x4 64-QAM)
Area Efficiency @ 25 dB SNR [Mbps/KGE]	4.06	0.32	0.18	0.55	15.6	13.1	28.1

4.13 Summary

In this chapter depth-first sphere decoding algorithm for MIMO detection has been studied. As the first step a configurable depth-first sphere detector architecture based on exhaustive search enumeration has been introduced. It has been shown that in M-QAM constellation space this architecture can support 16-QAM, QPSK and BPSK modulation schemes by masking the unwanted metrics. Configurability for antenna configurations is

[1] BER performance measures are defined in Table 3.1

achieved by applying modular buffer architecture for data buffer and using an expandable hierarchical state machine for buffer control.

In section 4.6 challenges of expansion of depth-first decoding algorithm to higher order modulation schemes have been explored. It has been shown that due to the sequential nature of depth-first sphere decoding algorithm, number of algorithm iterations increases dramatically at higher order constellation schemes such as 64-QAM modulation. The aforementioned challenge is inherent in depth-first decoding and can only be overcome by applying adjustments to the original algorithm. On the other hand we face hardware implementation challenges associated with memory size and the maximum achievable frequency. These two factors directly influence required silicon area and throughput of decoder and should be addressed by a joint effort in algorithm and architecture optimization.

Depth-first-K-Best (DFKB) sphere decoding algorithm has been introduced as an alternative to the strict depth-first sphere decoding. DFKB algorithm reduces the number of intermediate saved nodes by identifying and saving K child nodes with the smallest metrics. A level-based sphere radius (LBSR) definition accelerates node elimination process which eventually improves the latency of the algorithm.

The key to successful hardware implementation of DFKB sphere decoding with level-based sphere radius presented in section 4.7 is an efficient enumeration algorithm with the following features:

- The enumeration algorithm should have low gate count and propagation delay.
- It should be easily configurable for different modulation schemes.
- The enumeration algorithm should be able to enumerate K constellation points with the smallest node metrics that satisfy the sphere constraint test.

In section 4.8 *LCCE* enumeration algorithm that satisfies above conditions has been introduced. *MECU* architecture proposed based on *LCCE* algorithm performs all the required processing tasks on a parent node and enumerates eight child nodes as defined in DFKB algorithm.

In sections 4.9, 4.10 and 4.11 *MECU* block has been applied in implementation of MIMO detectors. DFKB-MECU has been introduced as low complexity high throughput architecture. The main drawback of this architecture is the dependency of throughput to

signal to noise ratio. We showed that adding an extra parallel detection engine in DFKB-MECU architecture reduces the throughput variations at the expense of adding more silicon area.

MECU block can also be used in implementation of fully parallel detectors. FSD-MECU architecture is an array processor with symmetrical structure based on *MECU* processing elements. FSD-MECU provides higher throughput compared to DFSD-MECU architecture with three times more gate numbers and at lower BER performance.

Chapter 5 Breadth-first tree search

5.1 Introduction

In breadth-first search and detection, detection algorithm computes and compares the metrics of all the nodes on one level of search tree before proceeding to the next level. There is no recursion in this algorithm and as a result pipelined architectures can be utilized for implementation of breadth-first detection algorithms.

Real-valued decomposition has been used extensively in the past in order to reduce the complexity of sorting networks at the expense of increasing number of pipelined stages. In a pipelined implementation of K-best algorithm, both throughput and bit-error-rate (BER) performance are dependent on the choice for parameter K. In order to be able to optimize the performance of detector under specific operating conditions, parameter K has to be adjustable.

Section 5.2 presents a review of K-best algorithm. The effects of channel and noise variations on performance of K-best detector are studied in section 5.3. In section 5.4, a configurable multi-mode pipelined architecture for real-valued K-best detector is presented. The number of pipeline stages in a complex-valued detector is N_t compared to $2N_t$ stages in a real-valued detector and as a result the overall silicon area of the complex-valued detector is smaller. In section 5.5 we show that by applying a 2-stage nod elimination/selection strategy and using *LCCE* enumeration algorithm discussed in section 4.8, the overall algorithmic and architectural complexity of complex-valued K-best detector can be reduced.

5.2 K-best algorithm

The received symbol vector $\mathbf{y}$ after preprocessing is mapped to $\mathbf{y}_q$ such that

$$\begin{aligned} \mathbf{y}_q &= \mathbf{Q}^{\mathbf{H}}\mathbf{y} \\ \mathbf{H} &= \mathbf{QR} \end{aligned} \tag{5-1}$$

Where $\mathbf{H}$ is the channel matrix, $\mathbf{Q}$ is an orthogonal and $\mathbf{R}$ is an upper triangular matrix. In K-best algorithm, a breadth-first tree search is conducted to search for the solution of detection equation (5-2); i.e. the detector visits all siblings of a node at one level of the search tree before it proceeds to the next level.

$$s_{ML} = \arg\min_{s\in\Lambda} \left\| \mathbf{y}_q - \mathbf{Rs} \right\|^2 \tag{5-2}$$

In this approach only K nodes which have the smallest accumulated metrics are saved and the rest are discarded. These selected K nodes will be saved as the parent nodes for the next level of tree expansion. After completing tree search, there will be K leaves with the smallest accumulated metrics. Each path in the search tree corresponds to one signal vector **s** and the path with the smallest metric is the detection result.

Expanding equation (5-2) yields equation (5-3).

$$s_{ML} = \arg\min_{s\in\Lambda} \sum_{i=1}^{N_t} \left| y_{qi} - \sum_{j=i}^{N_t} R_{ij} s_j \right|^2 \tag{5-3}$$

The node and branch metrics are defined respectively as:

$$\begin{aligned} T_i(\mathbf{P}_i) &= T_{i+1}(\mathbf{P}_{i+1}) + \left| e_i(\mathbf{P}_i) \right|^2 \\ T_{N_t+1}(\mathbf{P}_{N_t+1}) &= 0 \end{aligned} \tag{5-4}$$

$$e_i(\mathbf{P}_i) = y_{qi} - \sum_{j=i}^{N_t} R_{ij} s_j \tag{5-5}$$

The *node* metric $T_i(\mathbf{P}_i)$ is also known as *Partial Euclidean Distance* (PED). The search starts at level $i=N_t$ and progress towards the first level where $i=1$. In order to find the search results equations (5-4) and (5-5) have to be solved recursively. The partial symbol vector $\mathbf{P}_i(c_m)$ corresponds to a selection of transmitted symbols from level N_t to level i and can be described as:

$$\mathbf{P}_i(c_m) = [c_m, s_{i+1}, \ldots, s_{N_t}]^T \tag{5-6}$$

The transmit vector **s** corresponds to the complex constellation Λ containing P_c constellation points. As described in section 3.4.2, it is possible to transform the N_t-dimensional complex equation (5-2) to an equivalent $2N_t$-dimensional real-valued equation. The real operation reduces complexity of detection process since the number of child nodes per parent node in a real-valued constellation P_r becomes smaller than complex-valued constellation P_c ($P_c = P_r^2$).

The entries of equations (5-2) to (5-6) can be used for both real-valued and complex-valued constellation spaces. In case of real-valued constellation; it is assumed that the real-valued decomposition (RVD) as described in section 3.4.2 has been applied on

MIMO channel equation. The complex-valued K-best decoding algorithm is outlined in Figure 5.1.

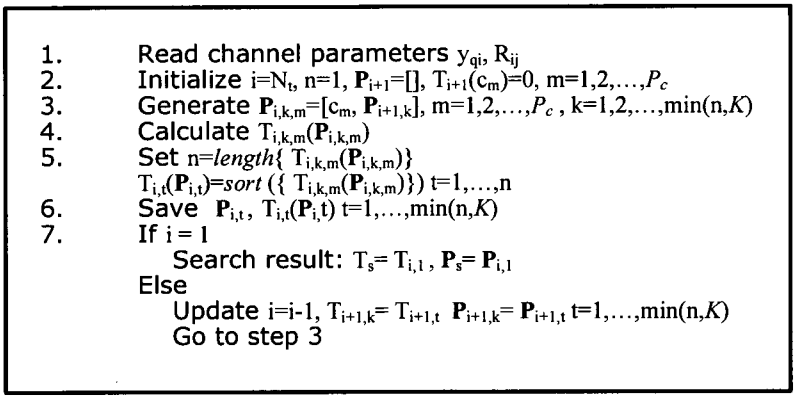


Figure 5.1 K-best detection algorithm

5.3 MIMO Channel Variations

In a MIMO wireless system, as wireless channel condition and noise level change, the receiver estimates channel parameters and signal to noise ratio (SNR) in order to optimize detection process. In this section we consider two different scenarios for channel variation and will analyze the effect of parameter K in each of these cases.

Figure 5.2 shows simulated BER performance in a K-best MIMO detector for two different channel conditions. The assumption here is that SNR remains unchanged. We assume that MIMO detector is initially operating at point 1. Due to channel variations, the operating point of detector moves to point 2 and as a result BER is increased. It is possible to improve BER by increasing K and move operating point from point 2 to point 3. The drawback of increasing K in a K-best detector is that throughput will be decreased.

In Figure 5.3, it is assumed that detector is required to operate with a fixed BER. Due to variations in SNR, the operating point moves from point 1 to point 2 and as a result BER increases. In order to maintain the required level of BER, detector should increase K at the expense of throughput.

The second scenario depicted in Figure 5.3 is related to the situation where channel conditions and SNR level have not changed but due to a change in operational requirements, receiver should improve its BER performance by increasing K (point 1 to 4) [61].

We can conclude that in order to optimize the performance of a K-best MIMO detector, detection parameter K should be run-time adjustable to improve the performance under different operating conditions. In the next section, we propose a VLSI architecture that satisfies this requirement.

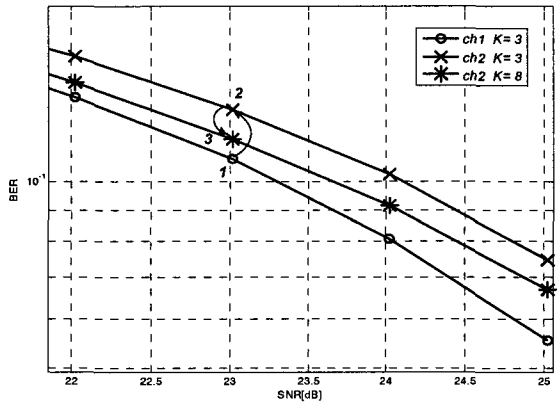


Figure 5.2 Effects of K and channel variations on BER performance

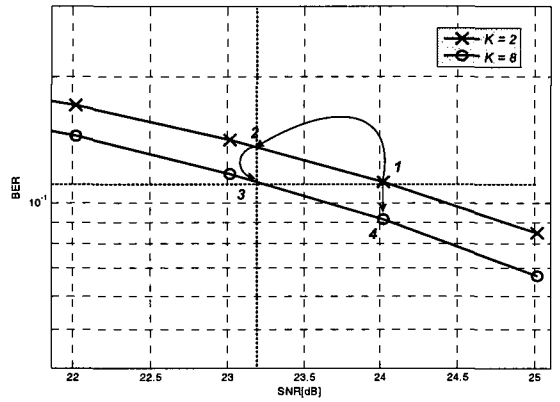


Figure 5.3 Effects of K and SNR on BER performance

5.4 Multi-mode configurable real-valued K-best architecture

For a real-valued constellation space, the pipelined architecture is composed of $2N_t$ processing elements (*PE*s) as shown in Figure 5.4. In this architecture each *PE* is associated with one level of a real-valued search tree. The i-th *PE* receives K data vectors from the preceding processing element (*PE* (i+1)). The entries of data vectors contain the previous vector symbols of the admissible nodes and their associated PEDs [63].

The task of each of the processing elements (*PE*) is to expand each of the K parent nodes to all their child nodes, calculate associated metrics for each of the child nodes and to

identify a set of K child nodes which have the lowest metrics (PEDs) and pass on the relevant information to the next processing element (PE) as illustrated in Figure 5.5. In the following sub-sections the detail design of processing elements will be discussed.

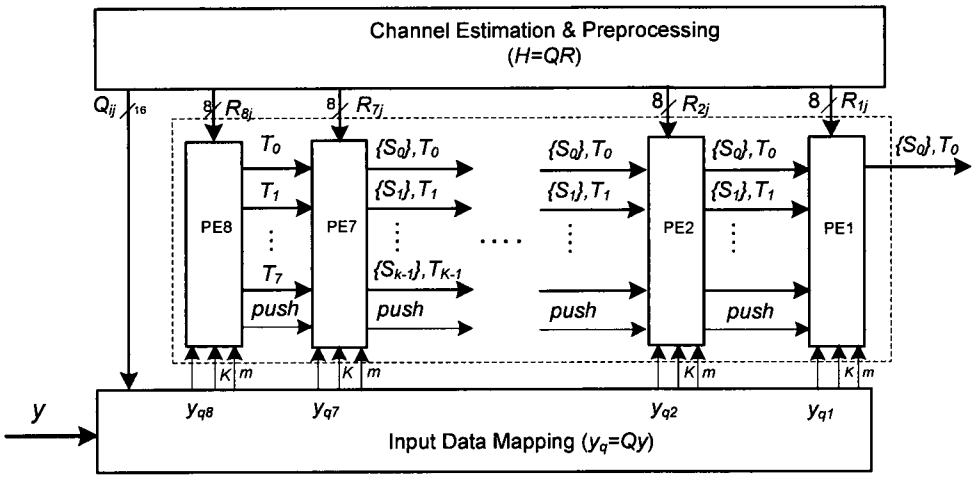


Figure 5.4 Pipelined architecture for 4x4 K-best detector

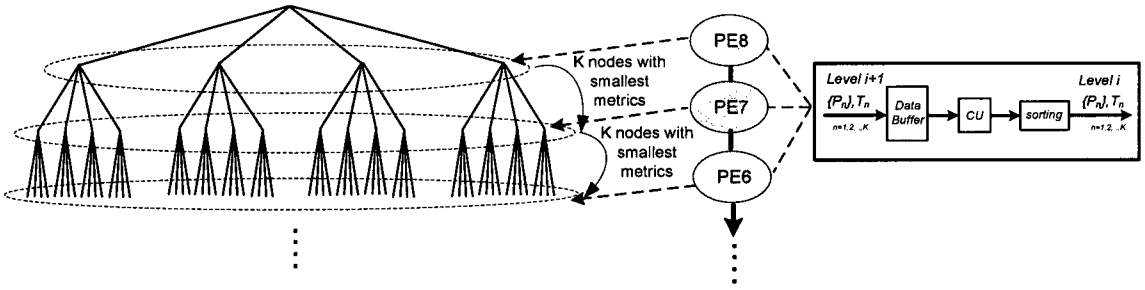


Figure 5.5 Processing element in pipelined architecture

5.4.1 Sorting Network

Sorting algorithms in general can be divided into parallel and serial categories. In parallel sorting all the inputs to the sorting network are processed simultaneously in a parallel interconnected network where as in serial sorting the inputs arrive serially to the sorting circuit in such a way that the new value to be sorted is inserted in an already sorted list. The drawback of the serial sorting circuit is the extra clock cycles required to process the incoming new values. Parallel-based sorting networks on the other hand are faster than serial sorting architectures but require more logic resources.

In the processing element of a real-valued K-best MIMO detector, on every clock cycle P_r new metrics (P_c new metrics in case of complex-valued detector) are computed. This

new set of metrics has to be compared and merged with the sorted list of metrics generated on the previous cycles. The process of sorting and merging repeats K times inside each processing element before data is passed to the next processing element in the pipeline. Due to this characteristic the inputs to the sorting network are neither fully parallel nor fully serial and as a result a sorting network with both properties provides the optimum result in terms of gate count and number of logic layers.

In Table 5.1, complexity of three parallel sorting algorithms i.e. Batcher's odd-even, Bitonic and Bubble algorithms for a 128 input network have been compared. It is also assumed that the incoming data are grouped in 16 set each comprised of 8 data value. Number of comparators shows the overall complexity of sorting network and number of vertical comparator layers is a measure of overall delay of the sorting network.

Table 5.1 Complexity of sorting algorithms

Sorting algorithm	Number of comparators	Number of vertical comparator layers
Parallel Batcher's odd-even	1471	28
Parallel Bitonic	1792	28
Parallel Bubble	8128	128
Pipelined Batcher's odd-even with feedback register	109	15

Batcher's odd-even merge-sort algorithm [55] is a fully parallel sorting algorithm that merges two sorted sequences into one sorted sequence. Complexity and number of vertical layers in this algorithm is comparatively lower than the other parallel sorting algorithms.

In order to further reduce the complexity of parallel sorting networks, we propose a pipelined sorting architecture based on Batcher's odd-even merging blocks. Architecture of the proposed *Pipelined Batcher's odd-even sorting* network with feedback register has been shown in Figure 5.6. This architecture can support 64-QAM modulation in real-valued constellation and value of K can be configured to K = 3,4,..,16. The basic idea behind the *pipelined sorting* architecture is to incrementally add new metrics to the

sorting network and compare them with the previous K smallest metrics and only retain K smallest ones and eliminate the rest.

On every clock cycle P_r (P_r = 8 for 64-QAM modulation) new metrics $T^m(s_i)$ are generated by the metric computation block. This new set of metrics need to be first sorted in an external 8-input sorting network to generate a sorted list ($T^m(s_0)$~$T^m(s_7)$) before entering the pipelined sorting network. The sorted list ($T^m(s_0)$~$T^m(s_7)$) has to be compared and merged with the previously sorted metrics. Since the goal in K-best algorithm is to eventually select K smallest metrics; on every clock we only need to save the smallest K metrics for comparison with the incoming new metrics on the next cycle and the rest can be discarded. We can use this property to further simplify the architecture of the sorting network by using a set of *feedback registers (FR)* to save K selected entries of the previous comparisons and reduce the size of the sorting network.

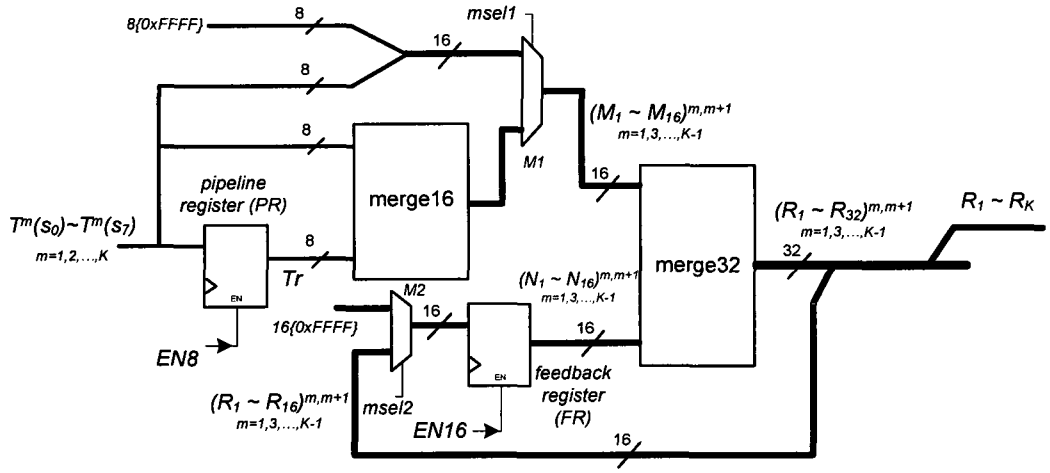


Figure 5.6 Pipelined sorting network

Pipeline register (PR) in Figure 5.6 captures $T^m(s_0)$~$T^m(s_7)$ for m=1,3,...$2n$-1 ($2n$-1 < K) in order to be merged with new set of $T^m(s_0)$~$T^m(s_7)$ on the next clock cycle. *EN8* signal toggles on every other clock cycles and forces pipeline register *PR* to only register $T^m(s_0)$~$T^m(s_7)$ when the index m is odd. Figure 5.7 depicts the timing diagram of signals in pipelined sorting network for two different cases: K=10 (even number) and K=9 (odd number).

The role of the merge blocks is to merge two sorted sequence into one sorted output sequence. Merge blocks are designed based on Batcher's odd-even merging algorithms

[53]. In *merge16* block, two sequences each comprised of eight elements are merged into a sixteen-element sequence.

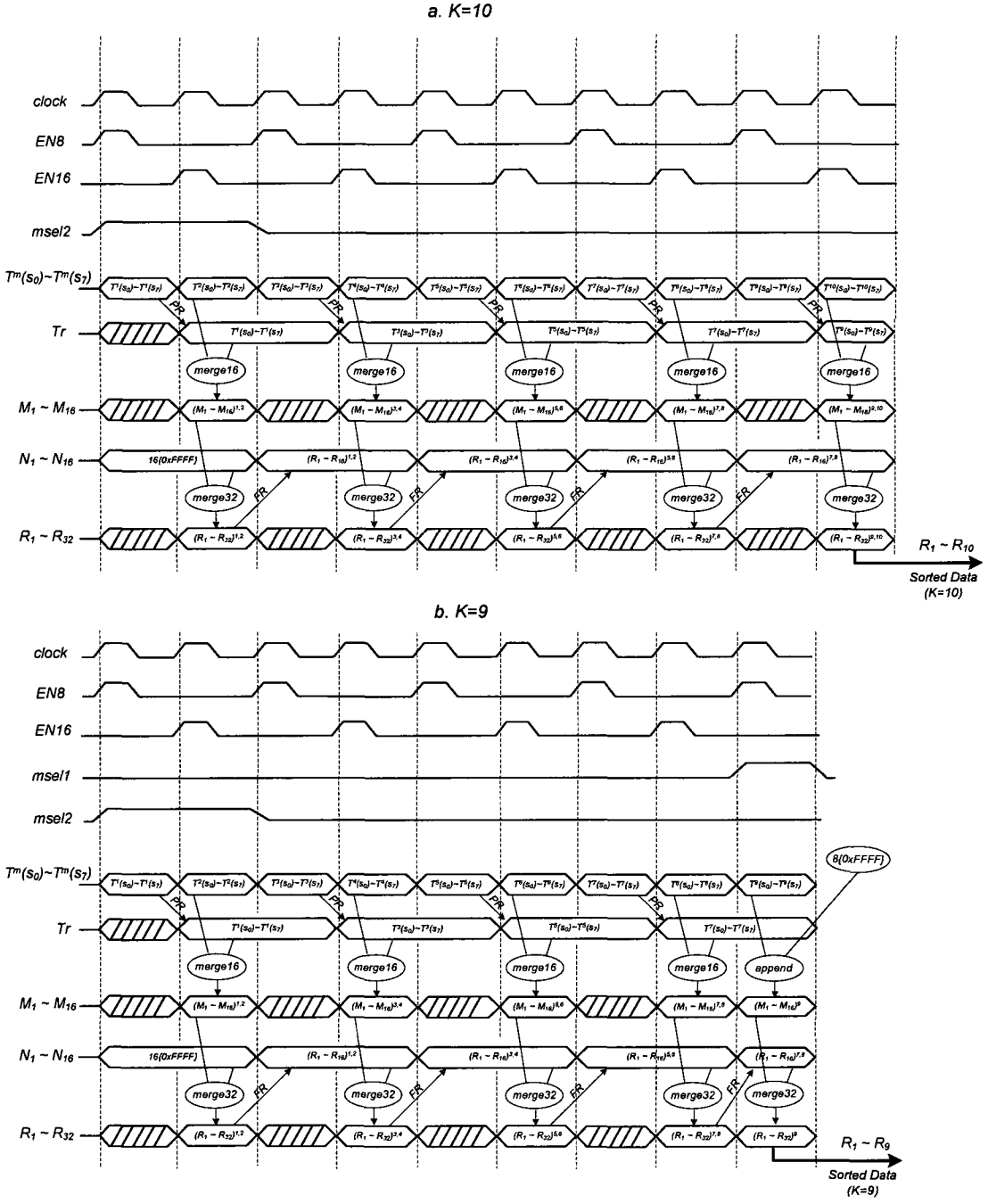


Figure 5.7 Timing diagram for $K = 10$ and $K = 9$

In Figure 5.8, architecture of *merge16* block is depicted. The basic building block in this architecture is a simple compare/exchange (*cmpx*) cell. The two inputs of *cmpx* cell are compared; the smaller input is connected to output '*l*' and the larger one is connected to

output '*h*'. The compare and exchange *cmpx* cell is used to build *merge4*, *merge8* and *merge16* blocks as shown in Figure 5.8. Similar to *merge16*, *merge32* block has two sorted 16-input sequences as inputs and merges them to generate a 32-element sorted list. The 16-element sorted list generated by *merge16* block $(M_1 \sim M_{16})^{m,m+1}$ is compared and merged with the sixteen best result of the previous comparison $(N_1 \sim N_{16})^{m,m+1}$ in *merge32* block. The best *sixteen* elements of the 32-element sorted list are saved in *feedback register (FR)* on the next clock cycle. *EN16* signal is the enable signal for *feedback register (FR)*.

Multiplexer *M2* initializes *feedback register (FR)* to *16{0xFFFF}* at the start of every sorting operation and as a result $(M_1 \sim M_{16})^{1,2}$ will be pushed to the top of the sorted list. When *K* is an odd number, the last group of metrics $(T^K(s_0) \sim T^K(s_7))$ are appended with *8{0xFFFF}* to generate a 16-element list and are then passed directly to *merge32* block from the upper path of *M1* multiplexer.

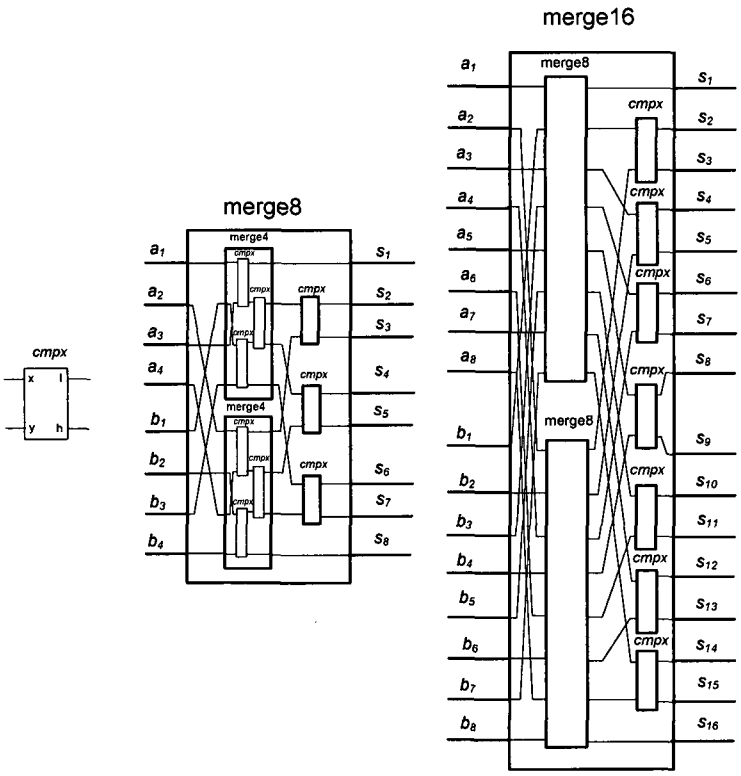


Figure 5.8 Architecture of *merge16* block

In Table 5.1 the complexity of *pipelined sorting* network has been compared to parallel architectures. It can be seen that both number of vertical comparator layers and number

of comparators have been reduced in this architecture compared to other parallel networks.

5.4.2 Processing element

In MIMO systems, the modulation modes of each of the transmitting streams can be set independently. As an example in Figure 5.9, the real valued tree for a 2x2 QPSK/16-QAM MIMO system has been shown. In this example, $\mathbf{S_1}$ is a QPSK modulated signal transmitted from the first antenna and $\mathbf{S_2}$ is modulated with 16-QAM modulation and transmitted from the second antenna.

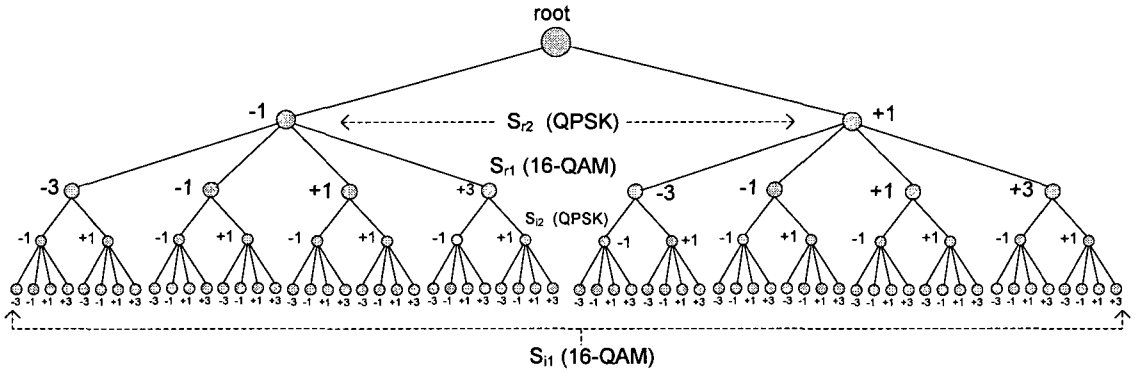


Figure 5.9 A 2x2 QPSK/16-QAM MIMO System

In such cases, each of the processing elements in Figure 5.4 should have the ability to reconfigure to the appropriate mode of modulation. Figure 5.10 shows architecture of the proposed configurable *PE* block. All candidate vector symbols and their associated metrics (T_n) are first stored in *Data Buffer* block.

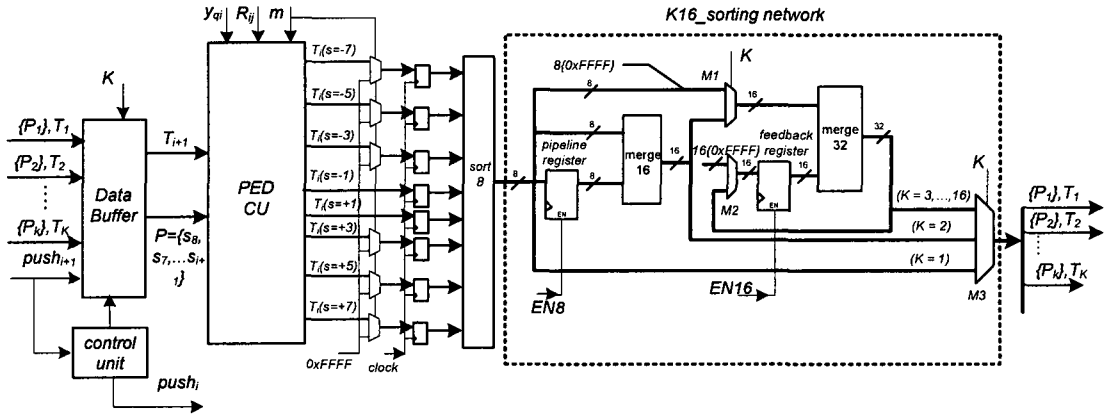


Figure 5.10 Architecture of *PE* block

Each metric value from the previous levels can generate up to eight new PED values depending on the type of modulation scheme (64-QAM, 16-QAM or QPSK), set by the modulation mode vector *m*. On every clock cycle a new metric value for one of the selected nodes from the previous level is read from *Data Buffer* and passed to the metric computation block.

PED Computation Unit in Figure 5.11; computes PEDs of all the children of a parent node. This block is designed such that it always calculates PEDs of child nodes considering 64-QAM modulation. The real constellation set in 16-QAM (({-3,-1,+1,+3}) and QPSK ({-1,+1}) modulation can be considered as a subset of constellation set in 64-QAM ({-7,-5,-3,-1,+1,+3,+5,+7}) modulation. In case of QPSK and 16-QAM modulation, a set of multiplexers mask the values of unused PEDs and replace them with the maximum possible values in the selected data system.

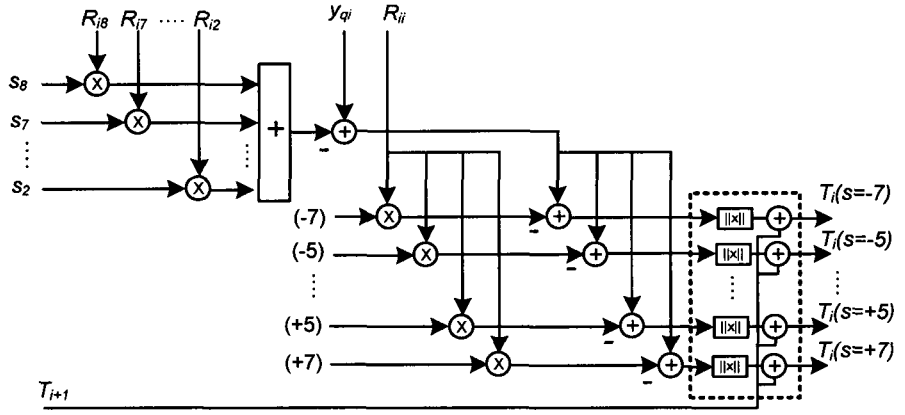


Figure 5.11 Architecture of PED computation block

Table 5.2 shows the values of modulation mode vector for different types of modulation schemes. The maximum set value depends on the number of data bits in the architecture; for a 16-bit fixed point system this value would be 65535 (0xFFFF). In Figure 5.10 *sort8* block moves the values for the masked PEDs to the end of the sorted queue and eventually drops them from the sorting queue.

The role of *K16_sorting network* in this architecture is to sort all the generated PEDs and select the smallest *K* metrics. In implementing *PE* block we are particularly interested in sorting networks that can be configured to support configurable *K* values. *K16_sorting* has been designed based on the pipelined sorting architecture presented in Figure 5.6.

The incoming metrics are first sorted in *sort8* block and a sorted list of metrics is generated for *K16_sorting* block. Multiplexer *M3* has been added to provide support for $K = 1,2$.

Control unit generates $push_i$ signal after $K+1$ cycles and writes the selected node metrics and partial symbol vectors to the next *PE*'s buffer. The output $push_i$ also signals the control unit of $(i\text{-}1)$th *PE* to start processing of a new set of data.

Each processing element can be independently configured for different modulation modes. Such configuration of processing elements can be used when each of the transmitting antennas uses a different modulation mode.

Table 5.2 Modulation mode vector

Modulation	Modulation mode vector *m*
QPSK	[1 1 1 1 1 1]
16-QAM	[1 1 0 0 1 1]
64-QAM	[0 0 0 0 0 0]

The BER simulation results for a 4x4 system with 64-QAM modulation and K=10 are shown in Figure 5.12.

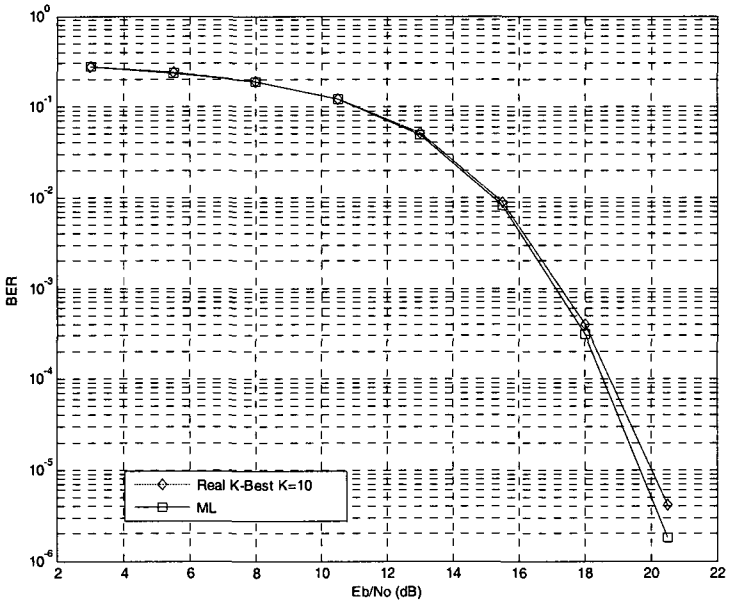


Figure 5.12 BER performance- Kbest real K=10, 64-QAM

5.5 Complex K-best decoder architecture

In section 5.4, a configurable architecture for implementation of processing element (*PE*) block based on real-valued *K*-best detection has been introduced. Number of pipelined stages in a complex *K*-best decoder is half of the number of pipelined stages in a real-valued based implementation and as results in complex-valued implementation the processing delay cycles can be reduced by 50%. Due to fewer number of processing elements in a complex-valued implementation, detector complexity (area) and power consumption is less than real-valued implementation.

The main implementation challenge for complex-valued K-best algorithm is the complexity of sorting network. As discussed earlier, in each processing element after computation of PEDs for all the child nodes of a parent node, PEDs have to be sorted to find the smallest *K* values. As we move to higher order complex constellations, i.e. 16-QAM and 64-QAM constellations, the complexity of the sorting block significantly increases due to exponential increase in the number of child nodes on every level. This issue will be discussed further in the following section.

5.5.1 Low complexity complex-valued K-best detection algorithm

K-best algorithm described in section 5.2 starts from level N_t, computes P_c branch metrics (P_c is the number of constellations in M-QAM constellation space Λ) and order them in increasing order and retains *K* nodes with the smallest metrics. At the next tree level, P_c metrics for each parent node from the previous tree level are to be computed. The total new metrics that has be sorted in this case is P_c^2 (if $P_c < K$) or $K \times P_c$ (if $P_c > K$) metrics and finally *K* nodes with the smallest metrics are saved for the next level and the rest of the nodes are discarded. The search continues to the first level of the tree by repeating this process. The node with the lowest metric at level one is the solution to MIMO detection problem. This process has been formulated in Figure 5.1.

In a 4x4 MIMO system with 64-QAM modulation on all four transmitting channels and considering $K = 10$, the total number of new generated metrics in complex-valued detector on every clock cycle would be equal to 640 entries compared to 80 entries in case of real-valued detection. In a *Pipelined Batcher's odd-even with feedback register* presented in Figure 5.6, this means that 64 new metric values enter the sorting network on every clock cycle compared to 8 metrics in real-valued detection.

In order to reduce the complexity of the sorting network one solution would be to break down the node selection process in two stages. Figure 5.13 shows the 2-stage K-best algorithm for a 4x4 64-QAM MIMO system. In this figure each level corresponds to one processing element that expands the parent nodes and selects the best child nodes to pass to the next level. The modified algorithm has two stages of sorting and pruning at every level except level N_t which has only one level of sorting to retain K nodes. In 2-stage K-best algorithm there are two parameters that influence the performance of algorithm, i.e. K_c and K which are associated with primary and secondary stage pruning respectively.

The selection process in 2-stage K-best algorithm can be best explained using the diagram in Figure 5.13 (K_c=8); starting at tree level N_t (N_t = 4 in the example shown in Figure 5.13), P_c new metrics are generated for each of the constellations c_m (m= 0,1,... 63) and sorted based on their respective values. This level has only one stage of pruning since the total number of nodes generated is less than 64 and can be sorted in one stage. If $P_c < K_c$ all the P_c nodes are saved in buffer for the next stage otherwise K_c nodes with the lowest metrics are first selected and then saved in the buffer. At level 3, K_c nodes out of P_c nodes for each of the survivor parent nodes from the previous level are selected and their metrics are computed in the first stage of pruning and selection. The first stage of pruning is followed by the second stage in which K x K_c (or P_c x K_c if $P_c < K$) new metrics from the first stage are compared in order to select the K nodes with lowest metrics and the rest are discarded. These K nodes are then saved in buffers and passed to the next stage. This process continues to level one in which the node with the lowest metric is selected as detected symbol.

The 2-stage elimination process reduces the complexity of the sorting network by limiting the number of nodes and consequently the number of metrics that have to be sorted in every cycle by the sorting network. The number of metrics to be sorted at each tree level in 2-stage pruning strategy is K x K_c ($K_c < P_c$) compared to K x P_c metrics in the complex K-best detector.

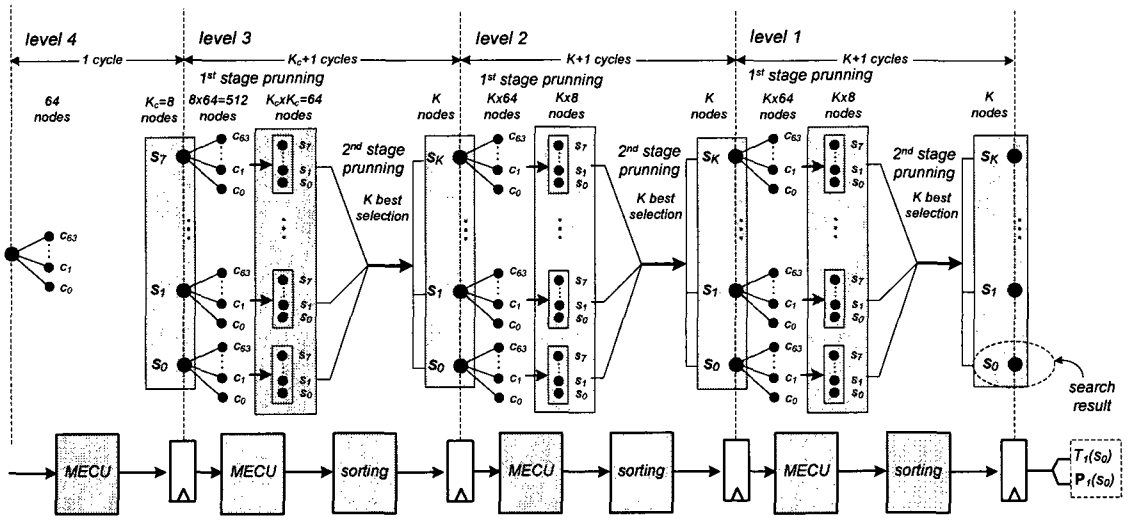


Figure 5.13 Low complexity K-best detection

The main challenge in practical implementation of complex K-best detector lies within the implementation of the sorting network. By following 2-stage sorting and selection strategy, the implementation challenge will be shifted to the first stage selection where K_c nodes with smallest metric from overall P_c child nodes have to be selected.

In section 4.7.2 Depth-First-K-Best (DFKB) algorithm has been introduced as a low complexity sphere decoding algorithm in which K child nodes with smallest metrics are selected and saved for further processing. In order to resolve the practical challenges of implementation of this algorithm, *LCCE* enumeration algorithm has been introduced in section 4.8 for enumeration of up to eight child nodes of a parent node with the lowest node metrics. *MECU* processing element has been introduced in section 4.8.2 as the hardware implementation of *LCCE* algorithm. The selection and elimination problem in the first stage pruning of 2-stage K-best algorithm is similar to the enumeration problem in DFKB algorithm and consequently *LCCE* algorithm can be used as a low complexity solution for this problem.

Pipelined implementation of 2-stage K-best algorithm for a 4x4 MIMO system is shown in Figure 5.13. Each pipelined stage includes one *MECU* block for the first stage pruning and a *sorting* block for the second stage pruning and selection. Number of pipelined stages in this detector is equal to the number of transmitting antennas N_t. At level 4 of the tree K_c nodes are generated and passed to level 3. In the following levels, first stage of elimination is performed using *MECU* block to select K_c=8 nodes from each of the parent

nodes. In the next level 8xK nodes are sorted to select the K nodes with the lowest metrics. At level one the partial symbol vector $\mathbf{P}_1(s_0)$ is the search result and solution to MIMO detection.

The BER performance plots of complex 2-stage K-best algorithm with K=10 and K_c=8, complex K-best algorithm with K=10 and ML detector for 64-QAM modulation are illustrated in Figure 5.14. As it can be observed in this figure the performance loss of 2-stage K-best is less than 0.2 dB compared to complex K-best algorithm and less than 1.4 dB compared to ML detector.

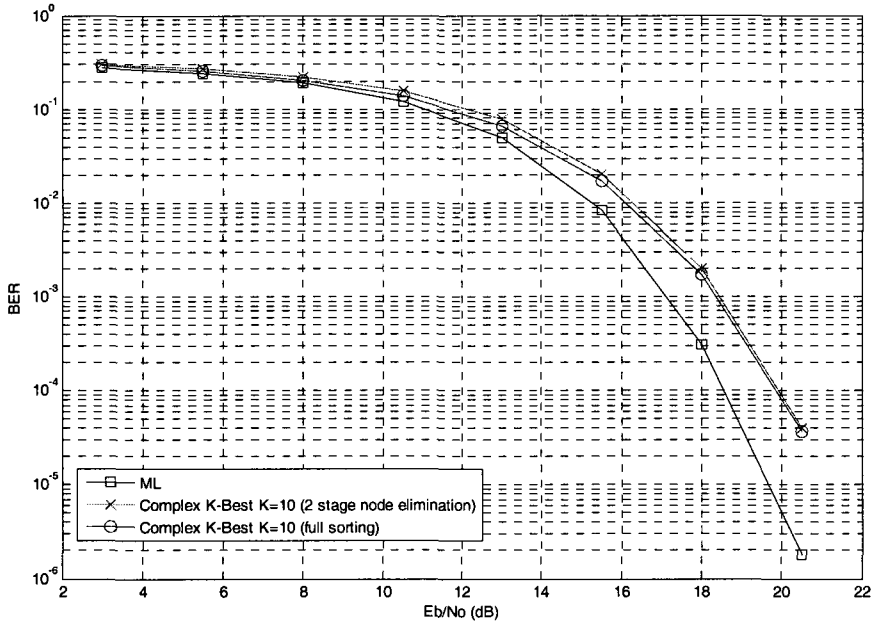


Figure 5.14 BER performance –complex K-best vs. 2-stage complex K-best algorithm

5.5.2 VLSI architecture for configurable complex-valued K-best detector

The 2-stage node pruning and selection algorithm introduced in section 5.5.1 enables design of high throughput, low latency and low area breadth-first MIMO detectors. The main benefit of 2-stage node elimination is the complexity reduction in *sorting* block which results in both higher clock frequency and lower silicon area for the detector. Higher clock frequency increases throughput and reduces the overall latency of the detector.

Top-level architecture of complex-valued detector for a 4x4 MIMO system is illustrated in Figure 5.15. This architecture consists of four pipelined stages, where each stage corresponds to one level of complex constellation tree representing one of the transmitting antennas. The first pipeline stage corresponds to level four of search tree consisting of a single *MECU* block that can produce up to eight node metrics $\{T_4(s_m),\mathbf{P}_4(s_m)\}$ (m=0,1,...7) where $T_4(s_m)$ and $\mathbf{P}_4(s_m)$ are PED and partial symbol vector for node s_m. The other pipeline stages consist of processing elements (PE_3, PE_2, PE_1) for computing updated node metrics, sorting and selection of K child nodes. Each processing element passes the metrics and partial symbol vectors ($\{\mathbf{P}_j,T_j\}$ j=0,1,..K-1) of the K selected nodes to the next processing element in the pipeline.

The incoming received symbol $\boldsymbol{y}$ is processed in preprocessing block and mapped to $\boldsymbol{y}_q$ according to equation (5-1) and enter the first pipelined stage. $\mathbf{R}$ is an upper triangular channel matrix also defined by equation (5-1). The *modulation mode* $\boldsymbol{m}$ signal specifies the modulation scheme for each of the transmitted symbols and is used to reconfigure each of the processing elements in the detector architecture. The resulted detected symbol is the output of the last processing element (PE_1) and is represented by $\mathbf{P}_{sol}$.

The core of each processing element (*PE*) in this architecture is one *MECU* processing unit introduced in section 4.8. *MECU* block is capable of undertaking two main tasks in the processing element. The first task is enumerating and preliminary sorting of selected child nodes which is followed by the second task which is computation of updated metrics of each of those nodes. One of the advantages of using *MECU* architecture is that the node metrics are already sorted and there is no need for an additional sorting block to select K_c nodes as discussed in section 5.5.1. Each *MECU* can be reconfigured to support BSPK/QPSK/16-QAM and 64-QAM modulation schemes. There is one pipeline stage between each *MECU* and *K16_sorting* network to reduce the length of the critical path.

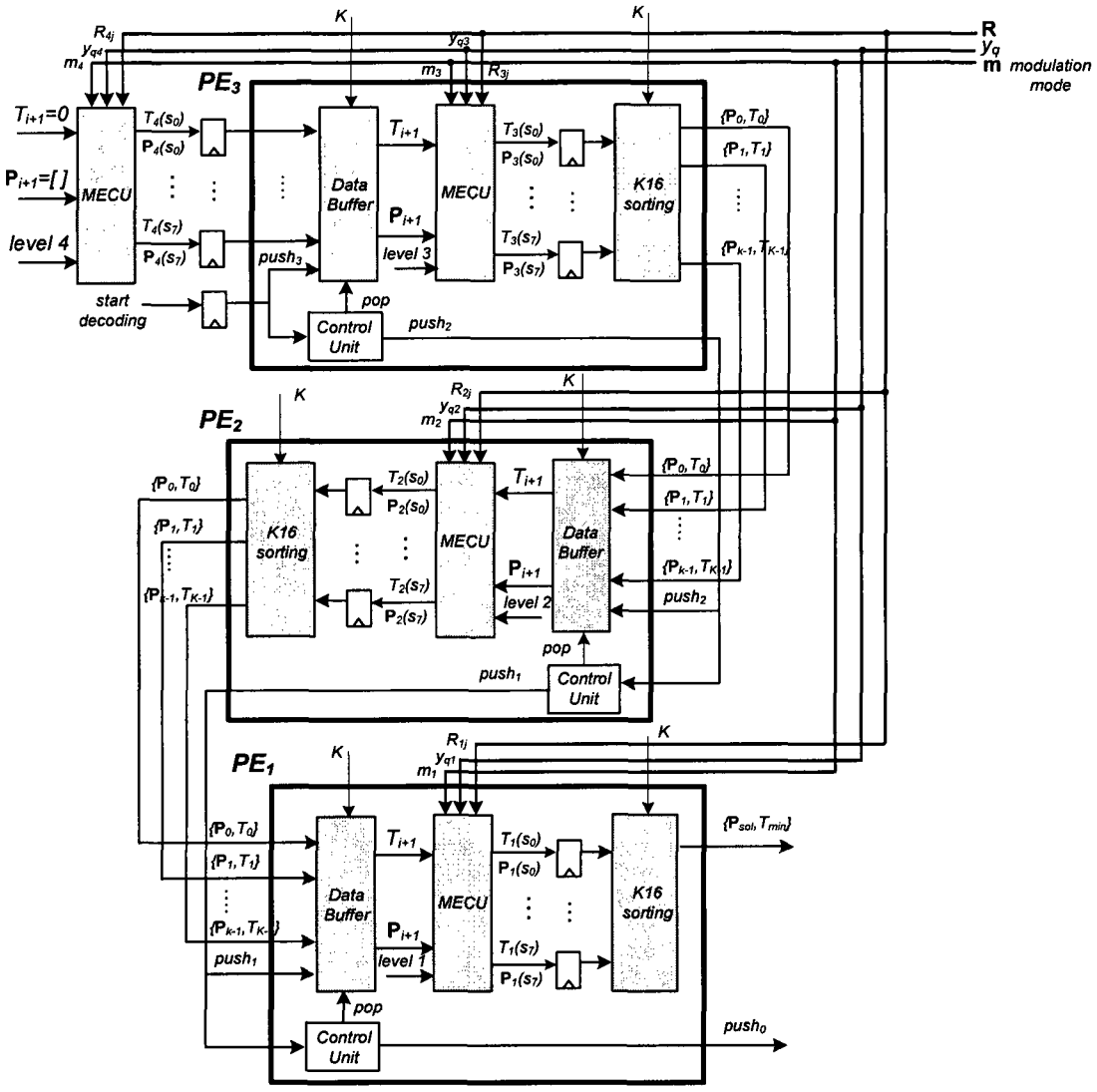


Figure 5.15 Complex-valued K-best decoder architecture

K16_sorting network performs the second stage of sorting and pruning. On every clock cycle K_c partial symbol vectors and metrics ($\{T_i(s_m),\mathbf{P}_i(s_m)\}$ i=3,2,1 m=0,1,…,K-1) enter *K16_sorting* which have to be merged with the previously sorted list of the metrics in order to generate the updated sorted list of K best metrics. The architecture of *K16_sorting* has been depicted in Figure 5.6. K-best detector parameter (K) can be changed to improve detector throughput or BER performance.

Partial symbol vectors and PEDs for the K selected nodes from the previous processing element ($\{T_i(s_m),\mathbf{P}_i(s_m)\}$ i=3,2,1 m=0,1,…,K-1) are saved in *Data Buffer* block. A *push*

signal from previous stage *PE* saves the data in *Data Buffer* and a *pop* signal asserted by *Control Unit* reads a new parent node from *Data Buffer.*

After K cycles, *Control Unit* asserts a *push* signal to the next stage *PE* and restarts reading data from *Data Buffer* block.

5.6 Implementation results

Based on the proposed architectures for real and complex valued K-best detector in sections 5.4.2 and 5.5.2, a 4×4 (QPSK/16-QAM/64-QAM) MIMO detectors has been designed, synthesized and mapped to 0.13 μm CMOS standard cell library. All the arithmetic computations are based on 16-bit fixed-point numbers. Data buffers have been implemented using registers to increase the speed at the expense of silicon area. The latency of each *PE* cell is K+1 cycles for both real- and complex-valued architectures. Table 5.3 shows performance figures for the proposed architecture and compares them with some of the reported real- and complex-valued architectures for K-best algorithm. As it can be seen detector parameter K which determines the BER performance of the detector has a considerable impact on throughput of the architecture. A larger K increases delay of the processing elements which in return reduces the throughput as defined by

$$DR = \frac{f_{clk} \cdot N_t \cdot \log_2^{P_c}}{K+1} \tag{5-7}$$

On the other hand, the maximum operating clock frequency of the circuit decreases since number of vertical logic layer increases in the sorting block for larger K values. The proposed MIMO detectors in this section have been designed with the aim of supporting variable Ks as well as three different modulation schemes for each antenna and as a result there is an additional hardware cost for both silicon area and timing associated with the added hardware flexibility.

Comparison of real- and complex-valued designs in Table 5.3 also reveals that we were able to improve latency, gate count and throughput of the K-best detector by applying 2-stage complex valued K-best algorithm. Reduction of the gate count and latency is mainly due to the fact that in complex-valued implementation the number of processing elements have been reduce to half of the real-valued implementation and as a result the overall silicon complexity has been reduced by 53%.

Table 5.3 Comparison of K-best implementation results

Architecture		Guo [16]	Guo [16]	Wenk [36]	Chen [65]	Shabany[71]	This work KBest-real	This work KBest-MECU
Number of Antennas		4×4	4×4	4×4	4×4	4×4	4×4	4×4
Modulation		16-QAM	16-QAM	16-QAM	64-QAM	64-QAM	64-QAM, 16-QAM, QPSK	64-QAM, 16-QAM, QPSK, BPSK
Detector algorithm		K-best	K-best	K-best	Relaxed K-best	K-best	K-best	K-best 2-stage elimination
Hard/Soft Decision		hard	soft	hard	hard	hard	hard	hard
Real/Complex Detector		real	real	real	complex	real	real	complex
Detector parameter- K		5	5	10	64	10	1~16	$K = 1\sim16$ $K_c = 8$
Tech [μm]		0.35	0.13	0.25	0.13	0.13	0.13	0.13
Equivalent gate number (KGE)		91 K	97 K	110 K	280 K	114 K	230 K	107 K
Clock Frequency (MHz)		100	200	52	270	282	80	140
Latency (cycles)		240	240	89	SNR dependent	168	14 ~ 119	7 ~ 52
Throughput (Mbps)	64-QAM	-	-	-	7.6~8.5	655	112 ~ 960	197 ~ 1680
	16-QAM	54	107	83	-	-	75 ~ 640	131 ~ 1120
	QPSK	-	-	-	-	-	37 ~ 320	65 ~ 560
Area Efficiency [Mbps/KGE]	64-QAM	-	-	-	0.03	5.9	0.5 ~ 4.2	1.8 ~ 15.7
	16-QAM	0.58	1.1	0.6	-	-	0.3 ~ 2.8	1.2 ~ 10.5
	QPSK	-	-	-	-	-	0.2 ~ 1.4	0.6 ~ 5.2

The increase in maximum clock frequency can be attributed to two factors. *MECU* processing block is a low complexity block and has a shorter critical path compared to the processing element designed for the real-valued implementation. It should also be mentioned that *MECU* directly operates on complex numbers. The second reason for the higher frequency in complex-valued implementation is the integration of one level of sorting into *MECU* block which eliminates the vertical layers associated with the *sort8* block in real-valued architecture in Figure 5.10. Area efficiency numbers are defined as the ratio between maximum throughput and equivalent gate number. The parallel/complex approach to sorting and metric computation improves the system throughput and provides better area efficiency in terms of throughput per gate number.

5.7 Summary

In this chapter we proposed high throughput configurable VLSI architectures for MIMO detection using K-best detection algorithm. The real-value implementation relies on real-valued decomposition of MIMO system model. A configurable pipelined implementation of this algorithm has been presented.

The sorting network is the main challenge in implementation of the processing elements in K-best detectors. Parallel sorting networks have less delay and higher throughput compared to serial sorting algorithms. Our solution for sorting networks reduces the complexity of parallel sorting networks by selecting the best K results on every clock cycle and eliminating the rest of the metrics. It also supports a variable range of Ks within the same architecture. Support for multiple modulation modes has been added to the processing element of the pipelined architecture by masking the unused real-valued constellation points.

The pipelined chain in a real-valued K-best MIMO detector is twice longer than the chain in a complex-valued detector. The main challenge in implementation of a complex-valued detector is dealing with the issue of exponential growth in the number of child nodes for every parent node. In 64-QAM modulation the sorting network has to process 64 new metrics on every clock cycle. The 2-stage K-best algorithm is an alternative way of reducing the complexity of complex K-best detector.

In this approach, the first stage of sorting in each tree level reduces the number of child nodes from P_c to K_c nodes. The K_c nodes are selected based on SE enumeration criteria, i.e. the nodes with smaller metrics have higher priority. The enumeration process for selection of K_c nodes can be accomplished in one clock cycle. After the first stage of node elimination, the total number of child nodes reduces to K_cK compared to P_cK in K-best algorithm. The first stage of node elimination can be accomplished using *LCCE* enumeration algorithm proposed in section 4.8.1. In this algorithm eight child nodes with the smallest metrics are selected without explicitly computing and sorting the metric values. The computation of PEDs for the selected nodes is implemented in parallel to the enumeration process. As showed in section 4.8.1, the proposed enumeration algorithm is also configurable and can support different modulation schemes. We proposed a configurable complex-valued MIMO detector that supports 64-QAM, 16-QAM and

QPSK modulation modes over a variable range of K values. The complex-valued architecture proposed in this chapter substantially improves the throughput, latency and area compared to the configurable real-valued implementation and other implementations proposed earlier in the literature.

Chapter 6 Detector Implementation, Verification and Emulation

6.1 Introduction

Detector design process starts with algorithm exploration by simulating key performance figures such as throughput and bit-error-ratio for each algorithm in Matlab environment. The selected algorithms are mapped to hardware architectures by following steps in design flow. The implementation and verification flow is composed of four main steps i.e., design and optimization, functional verification, hardware emulation or silicon implementation.

In hardware architectural design phase, HDL models of detector subblocks are developed and integrated into detector top-level. Later HDL models of top-level are verified in Verilog test benches for testing the basic functionalities. The process of verification and optimization of detector cores are completed in Modelsim-Simulink test benches. Numerical precision is concerned with the number of significant bits in number representation and the number of bits assigned for integer and fractional parts. Modelsim-Simulink environment has also been used to optimize the number of bits for real and fractional parts.

VLSI circuit complexity numbers in the early stage of design are expressed in terms of equivalent gate number instead of square millimetres to eliminate the effect of technology scaling. For FPGA implementations; number of combinational logic elements and registers has been used for complexity comparison between different algorithm implementations.

The figures of merits (FOM) have been expressed in Mbps/KGE and Mbps/mW to express the efficiency of different architecture. The efficiency factor Mbps/KGE shows the average throughput achieved for every 1000 NAND gates in design and is a measure of area efficiency in design. The Mbps/mW factor signifies the throughput generated for one milliwatt of consumed power and is a measure of power efficiency in detector design.

In section 6.2 the functional verification environment for validation of detector designs in this dissertation will be introduced. The verification environment can be used for algorithm development, RTL verification, performance measurements, and optimization. In section 6.3, the hardware emulation platform using FPGAs is presented. In this platform, detector designs are implemented on Altera® FPGAs and tested with the test vectors generated by the MATLAB-Simulink wireless channel model. The silicon implementation has been discussed in section 6.4 for the DFKB-MECU architecture targeting TSMC 0.18 μm six-metal CMOS process.

6.2 Functional verification

Functional verifications of detector architectures have been performed using Verilog and *Simulink* based test benches. Verilog test benches have been used to verify the basic functionality of RTL designs. *Simulink* based test benches on the other hand provide a comprehensive verification environment for more advanced verifications as well as fixed-point performance measurements. *ModelSim* simulator has been used for RTL verifications.

Link for ModelSim is a cosimulation interface that integrates *MATLAB* and *Simulink* into the FPGA and ASIC design flows by providing a bidirectional link between *MATLAB*, *Simulink* and *ModelSim* HDL simulator. The cosimulation environment enables efficient and in depth verification of RTL level models from within *Simulink* environment. It can also be used for generating test vectors, debugging, and verifying HDL codes against their *Matlab* or *Simulink* models.

Simulink test benches model communication channel, transmitter and analog modules of the receiver and provide the link to the RTL codes of MIMO detectors. For the purpose of auto-checking and performance comparison, a Matlab model of MIMO detector algorithm is co-simulated with RTL design.

Figure 6.1 shows the co-simulation environment developed for verification of MIMO detectors. *Serial Random Data Generator* is the source of random data in this verification test bench which is mapped into N_t parallel data streams. The serial data stream after *Multiplexing* and *QAM Modulation* is transmitted over *MIMO channel* to the receiver side of the test bench.

Receiver side of the test bench includes *Matlab* model of detection algorithm connected in parallel with HDL model of detector architecture implemented in Verilog HDL language. *MIMO Detector Matlab SR* model is the reference detection algorithm implemented using *Matlab level-2 S-functions*. The output of MIMO detector model provides a comparison reference for both verification and performance comparison purposes.

Modelsim Interface provides an interface from *Simulink Test bench* to the RTL design of MIMO detector running in *Modelsim* simulator. Due to different simulation speeds in *Simulink* and *Modelsim* simulators, a queuing mechanism is needed in order to synchronize the data transfer between the two simulation environments. *Modelsim* simulator requires variable number of clock cycles to process the incoming data while *Matlab* model processes the incoming data in one simulation cycle independent of the number of iteration through detection algorithm. The output of *MIMO Channel* block is stored in *Queue* using *push* signal which is asserted synchronously at symbol period rate. MIMO detector asserts *pop* signal when detection process for one set of symbols is finished and detector is ready to accept a new set of received symbols.

The received symbols from channel are in floating-point format which should be converted to fixed-point format for the purpose of RTL simulation and design. *Floating point to Fixed-point converter* block in the simulation environment depicted in Figure 6.1 performs this conversion task.

A pulse generator asserts *start* and initiates the decoding process in MIMO detector. The assertion of *ready* signal by MIMO detector shows the completion of detection process for one set of received symbols. In this case a new set of received symbols are popped out of the queue for processing in the detector. The serial data generated by the transmitter has to be delayed before BER calculations to compensate for detector processing delay. The average detection cycle is a measure of throughput performance of MIMO detector under test. The number of iteration cycles is averaged in *mean* block and is displayed as part of performance measurement results.

This verification test bench can also be used to study the effects of various design parameters on performance of MIMO detectors. The *scaling* of fixed-point conversion

influences both throughput and BER performance of the detector and can be optimized using the simulation test bench.

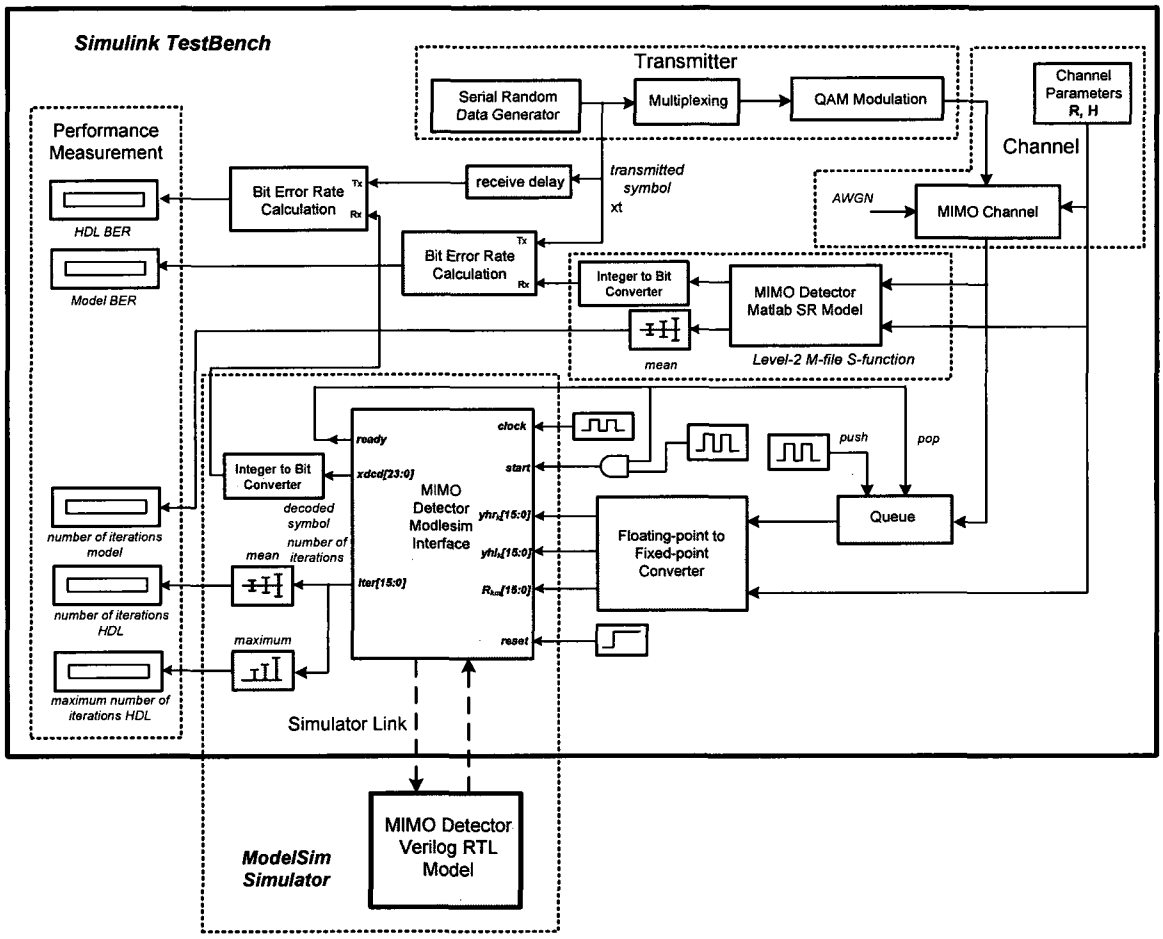


Figure 6.1 Simulink-Matlab-Modelsim co-simulation

6.3 FPGA design and prototyping

MIIMO detector designs in Chapter 4 and Chapter 5 have been synthesized and mapped on *Altera CycloneII* and *StratixIII* FPGA chips. *CycloneII* FPGA family is based on 1.1-V 90-nm, eight all-copper layers SRAM process technology with densities up to 68416 logic elements (LE) and up to 1152 Kbits of RAM [56].

StratixIII family is based on 1.1-V, 65-nm all-layer copper SRAM process technology and provides up to 135K *Adaptive Logic Modules ALM* (338K equivalent logic elements LEs) and up to 18381 Kbits of RAM memory [57]. In this family *Logic Array Block* (*LAB*) is composed of basic building blocks known as *Adaptive Logic Modules* (*ALMs*) that can be configured to implement logic, arithmetic, and register functions. Each LAB

consists of ten ALMs, carry chains, shared arithmetic chains, LAB control signals, local interconnect, and register chain connection lines. Each ALM contains a variety of look-up table (LUT) based resources that can be divided between two combinational Adaptive LUTs (ALUTs) and two registers.

Table 6.1 and Table 6.2 present resource utilization results as well as maximum achievable clock frequency for detector architectures implemented on *CycloneII* and *StratixIII* FPGA chips. *Quartus* synthesis tool has been used for FPGA synthesis, placement and routing.

Table 6.1 Synthesis Results – Cyclone II (EP2C70F672C6) FPGA

Decoder architecture	DFKB-MECU	DFKB-parallel	KBest-MECU	FSD-MECU		DFSD (reference)	KBest-real (reference)
				(6-1-1-1)	(8-1-1-1)		
Total Number of logic elements	6293 (9%)	11730 (17%)	19366 (28%)	30917 (45%)	39984 (58%)	12457 (18%)	50468 (74%)
Total combinational functions	6032 (9%)	11430 (17%)	19361 (28%)	30854 (45%)	39882 (58%)	12151 (18%)	50354 (74%)
Total Number of registers	1137 (2%)	1837 (3%)	2575 (4%)	1001 (1%)	1232 (2%)	1256 (2%)	6699 (10%)
Max. Freq. (MHz)	21	20	23	23	23	15	15
Throughput @ 25 dB SNR (Mbps)	96	96	32 ~ 276	552	552	33 @ (16-QAM)	21 ~ 180
Latency (μs)	0.25	0.25	0.3 ~ 2.2	0.21	0.21	0.33	0.9 ~ 7.9

Table 6.2 Synthesis Results – Stratix III (EP3SL70F484C2) FPGA

Decoder architecture	DFKB-MECU	DFKB-parallel	KBest-MECU	FSD-MECU		DFSD (reference)	KBest-real (reference)
				(6-1-1-1)	(8-1-1-1)		
Logic utilization	1%	2%	2%	8%	10%	2%	4%
Total combinational ALUTs	4654 (9%)	8927 (17%)	14544 (27%)	22261 (41%)	28593 (53%)	8094 (15%)	40661 (74%)
Total Number of registers	1138 (2%)	1842 (3%)	2575 (5%)	1007 (1%)	1247 (2%)	1258 (2%)	6700 (12%)
Max. Freq. (MHz)	38	38	43	40	40	30	29
Throughput @ 25 dB SNR (Mbps)	182	182	60 ~ 513	960	960	66 @ (16-QAM)	41 ~347
Latency (μs)	0.1	0.1	0.16 ~ 1.2	0.12	0.12	0.16	0.5 ~ 4.1

DFSD architecture synthesized for BPSK/QPSK/16-QAM system is based on exhaustive enumeration and can achieve a maximum throughput of 33 Mbps in Cyclone II FPGA in 4x4/16-QAM mode with close to maximum likelihood BER performance.

DFKB-MECU architecture utilizes MECU enumeration and computation block and improves maximum throughput of the detector and at the same time reduces the logic utilization factor. The logic utilization factor in DFKB-parallel architecture is almost twice larger than DFKB-MECU architecture due to the added search engine in this detector. The main advantage of this architecture as discussed in section 4.11 is the improved throughput in lower signal to noise ratios compared to DFKB-MECU architecture.

The FPGA synthesis results presented in Table 6.1 and Table 6.2 demonstrate that DFKB-MECU architecture has the lowest logic utilization compared to other architectures examined in this dissertation. KBest-real architecture on the other hand has the highest logic utilization factor in *CycloneII* FPGA mapping as well as the highest number of utilized registers.

KBest-MECU architecture is a complex K-Best detector that uses MECU block as the main processing unit. It can be observed that both the logic element utilization and register numbers in this architecture has been reduced by 60% compared to KBest-real architecture. In both KBest-real and KBest-MECU, parameter K is adjustable and as a result the maximum throughput performance is variable.

FSD-MECU architecture has the highest maximum throughput compared to the other architectures presented. The main drawback of this architecture as discussed in 4.10 is its high bit error rate compared to sphere and K-best detectors.

Hardware-in-the-Loop (HIL) simulation is a form of real-time simulation that is increasingly used in development and test of complex real-time embedded systems. The purpose of HIL simulation is to provide an effective platform for developing and testing real-time systems. In HIL simulation a real time hardware system is added to the simulation loop consisting of software models.

DSP Builder software tool provides a Simulink library for hardware in the loop co-simulation between Simulink and Altera FPGA boards. Adding Hardware-in-the-Loop (HIL) block to the Simulink test bench developed for verification of MIMO detectors allows us to co-simulate MIMO communication link (transmitter and channel) modeled using Simulink block sets with a physical FPGA board implementing detector design in one simulation-emulation environment. The detector design embedded in an FPGA runs

faster than RTL simulation and as a result provides a more efficient verification environment.

Figure 6.2 shows the block diagram of MIMO detector emulation environment implemented on *Altera DE2* development platform. *CycloneII* chip is mounted on a *DE2* development platform. A JTAG interface provides the communication link for data transfer between Simulink test bench and FPGA board. The emulation platform can be used for both verification and performance measurements.

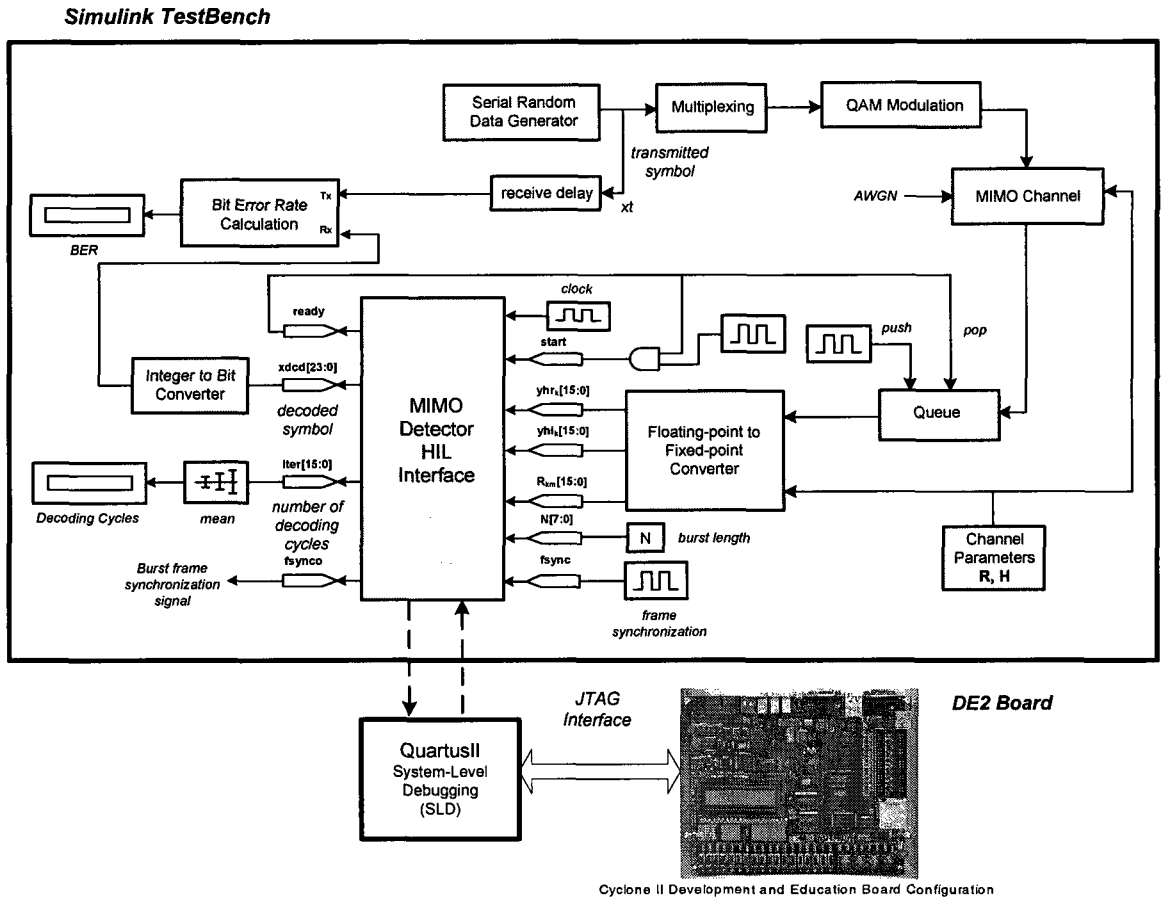


Figure 6.2 MIMO detector emulation platform

Similar to the test bench in Figure 6.1, Simulink models transmitter and MIMO channel. In order to synchronize data flow between Simulink and FPGA board a FIFO queue has been used. A pulse generator asserts *start* signal in order to initiate detection process. The assertion of *ready* signal by detector shows that decoding of one symbol set has been completed and a new set can be popped out of the queue for processing.

In order to further increase the simulation speed, *frame mode* of data transfer has been utilized to burst data between Simulink and FPGA board. In frame mode, the HIL block monitors *fsync* and *fsynco* signals and increases output delay in order to align output data frames with the input data frames. The burst packet size in frame mode is a multiple of frame packet interval. Parameter N in emulation environment defines the burst packet size.

The emulation setup shown in Figure 6.3 consists of a DE2 platform connected to the USB port of a laptop computer running the Simulink test bench. The USB port provides the serial link for JTAG interface to DE2 board.

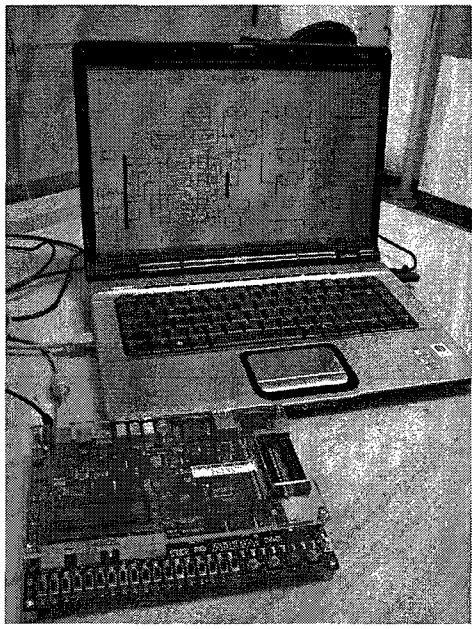


Figure 6.3 MIMO communication system emulation setup

6.4 Power analysis

Power consumption for different detector architectures introduced in this dissertation has been estimated based on 50% switching activity on all circuit nets and the results are listed in Table 6.3.

Dynamic power dissipation in every node in a CMOS circuit can be characterized by equation (6-1) where C_L is the node capacitive load, f_{clk} is the circuit clock frequency, V_{DD} is the supply voltage and α is the activity factor. The switching or activity factor is defined as the probability that in each clock cycle a 0 to 1 transition occurs at the node [40].

$$P_{dynamic} = \alpha C_L V_{DD}^2 f_{clk} \quad (6\text{-}1)$$

In MIMO detectors, increasing clock frequency increases both throughput and power consumption. The implementation results of detector architectures presented in Table 4.5 and Table 5.3 have been optimized for maximum detector throughput by maximizing clock frequency. In Table 6.3 the power efficiencies for different detector architectures have been defined as the ratio between the throughput and the consumed power.

Comparison of the power consumption results for DFSD and DFKB-MECU architectures reveals that depth-first detector based on MECU processing block has higher power efficiency and lower power consumption. This is due to both increased throughput and lower gate count in DFKB-MECU architecture.

Table 6.3 Comparison of power consumption

Decoder Architecture	DFSD	DFKB-MECU	PDFKB	FSD-MECU	KBest-real	Kbest-MECU
Antenna	$N_t = 2\sim4$ $N_r = 2\sim4$	$N_t = 2\sim4$ $N_r = 2\sim4$	$N_t = 2\sim4$ $N_r = 2\sim4$	4×4	4×4	4×4
Modulation	16-QAM, QPSK, BPSK	64-QAM, 16-QAM, QPSK, BPSK	64-QAM, 16-QAM, QPSK, BPSK	64-QAM, 16-QAM, QPSK, BPSK	64-QAM, 16-QAM, QPSK	64-QAM, 16-QAM, QPSK, BPSK
Detection Algorithm	Depth-first sphere decoder	Depth-first-K-best (DFKB) LB-radius	Parallel DFKB LB-radius	Fixed sphere decoder	Real-valued K-best	Complex-valued K-best 2-stage elimination
CMOS Technology	0.13 μm	0.13 μm	0.13 μm	0.13 μm	0.13 μm	0.13 μm
Clock Freq. (MHz)	120	140	140	150	80	140
Max. Throughput (Mbps) @ 25 dB SNR	264	672	672	3600	112 ~ 960 (K = 1~16)	197 ~ 1680 (K = 1~16)
Power (mW)	16.4	12.4	13.3	49.8	21.3	25.8
Power Efficiency (Mbps/mw)	16	54.2	50.5	72.2	5.2 ~ 45 (10 @ K = 8)	7.6 ~ 65.1 (14.4 @ K = 8)

The PDFKB architecture is the parallel implementation of DFKB-MECU architecture and the power consumption has been increased compared to DFKB-MECU. The main reason for lower power efficiency in the parallel PDFKB is that as previously discussed

in section 4.11, adding the second search engine has no impact on the throughput of the detector at high signal to noise ratio and only improves detector throughput at lower signal to noise ratios.

The fully parallel FSD-MECU architecture has the highest power consumption compared to the other detectors presented in this dissertation. Due to the high throughput of this architecture, the power efficiency of this detector is also the highest. Comparison of power consumption results for Kbest-real and Kbest-MECU reveals that the power consumption is almost in the same range but due to the higher data throughput of Kbest-MECU architecture, the power efficiency is also higher and out performs Kbest-real efficiency figures.

6.5 ASIC implementation

6.5.1 Design synthesis

A configurable 4x4/3x3/2x2 64-QAM/16-QAM/QPSK/BPSK MIMO detector based on DFKB-MECU architecture has been considered for ASIC implementation. Decoder design has been modeled in Verilog HDL at register-transfer level (RTL) and then taken through CMC's digital design flow for ASIC implementation. *Synopsys Design Compiler* has been used for synthesis and optimization of the RTL design into Artisan 0.18 μm CMOS standard cell library, targeting TSMC 0.18 μm six-metal CMOS process.

6.5.2 Silicon implementation

The synthesized gate level netlist provided by *Design Compiler* has been imported into *Cadence SOC Encounter* for layout design. *Cadence SOC Encounter* tool has been used for placement, power grid design, and clock tree synthesis and routing. *Virtuoso* has been used for LVS and final top-level connections. Design rule checks (DRC) have been completed using *Mentor Graphics Calibre*. Post-layout timing has been verified using *Synopsys Prime Time* with back annotated delays data. Figure 6.4 shows the layout of the DFKB-MECU detector chip. The total die area is 1.71 mm^2 and the detector core area is 0.87 mm^2.

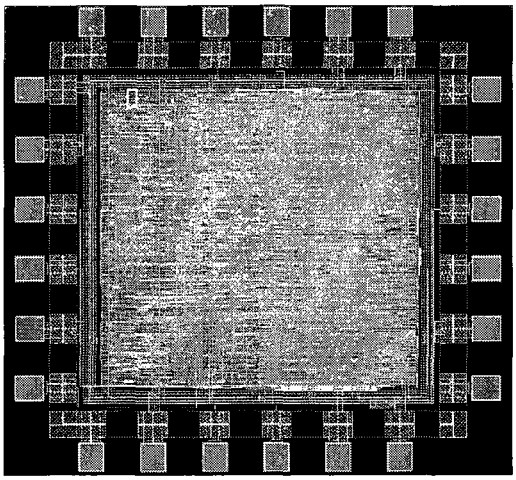


Figure 6.4 Layout of the routed chip

Table 6.4 compares DFKB-MECU implementation results with a number of previously reported MIMO detector ASICs.

Table 6.4 Comparison of ASIC design implementations

Decoder Architecture	**Chen [65]**	**Lin [66]**	**Huang [70] ASIC-I**	**Huang [70] ASIC-II**	**This work DFKB-MECU**
Antenna	4×4	4×4	$N_t = 2\sim4$ $N_r = 2\sim6$	$N_t = 2\sim4$ $N_r = 2\sim6$	$N_t = 2\sim4$ $N_r = 2\sim4$
Modulation	64-QAM	64-QAM, 16-QAM, QPSK	64-QAM, 16-QAM, QPSK	64-QAM, 16-QAM, QPSK	64-QAM, 16-QAM, QPSK, BPSK
Detection Algorithm	Relaxed K-best	SD-SIC	OSIC V-ML	OSIC V-ML	DFKB
CMOS Technology	0.13 μm	0.09 μm	0.13 μm	0.13 μm	0.18 μm
VDD (V)	1.2	1.1	1.1	1.1	1.8
Core Area (mm^2)	2.38	0.674	-	-	0.87
Chip (Die) Area (mm^2)	-	-	5.96	5.29	1.71
Clock Freq. (MHz)	270	166	55	71	60
Throughput (Mbps) @ 25 dB SNR	8.57 @17.7 dB	90	36	114	288
Estimated Power (mW)	94	58	12.8	30	45.4
Power Efficiency (Mbps/mW)	0.09	1.55	2.81	3.80	6.34

The architectures proposed by Huang *et al.* in [70] are configurable architectures supporting QPSK, 16-QAM and 64-QAM. In this approach, MIMO detector is divided into a core detector and a residual detector. The core detector is a 2x2 V-BLAST ML detector used for detection of the first two symbols. The residual detector is an ordered successive cancelation (OSIC) detector that detects the rest of the transmitted symbols. In ASIC-II architecture the performance of the detector has been improved by adding a parallel search in V-ML core detector and also by using pipeline stages between functional blocks. The detector proposed by Lin *et al.* in [66] is a reduced complexity sphere detector. In this approach the complexity of candidate searching has been reduced by shifting the center of constellation space with scalable radius.

Comparison of the core and chip areas of the DFKB-MECU design compared to other architectures presented in Table 6.4 shows that DFKB-MECU implemented in CMOS 0.18 μm technology has a lower silicon area compared to other implementations.

The high throughput in DFKB-MECU architecture is achieved at a comparatively low clock frequency. This is mainly due to the more efficient symbol detection algorithm employed in this architecture. The lower clock frequency and silicon area also leads to higher power efficiency in this architecture.

Chapter 7 Conclusions and Future Contributions

7.1 Conclusion

The major challenge facing future wireless communication networks is to provide high data rate wireless access to mobile users at high quality of service (QoS) without increasing the frequency spectrum. The link reliability in a wireless channel is affected by the fading caused by destructive addition of multipath components and interference from other users. Multiple-input multiple-output (MIMO) wireless technology meets these demands by offering increased spectral efficiency through spatial multiplexing and improved link reliability due to antenna diversity and space-time coding. These improvements on the other hand come at the cost of significant increase in complexity of signal processing algorithms in both transmitter and receiver systems. The computational complexity of MIMO receivers also increases dramatically with respect to the number of transmitter/receiver antennas as well as the type of modulation used for transmission.

Configurability is the new design paradigm in mobile baseband processing. Programmable general purpose processors and digital signal processors (DSPs) provide the level of flexibility required by the next generation wireless standards but the current technology advancement for these devices can not provide the level of computational performance and power consumption required by MIMO systems implemented in portable devices. In the next decade there will still be a gap between the algorithmic complexity and performance of programmable processors. A baseband processing system based on hardware accelerators and programmable processors can overcome the complexity challenges of the new algorithms and at the same time meet the configurability requirements of the next generation wireless systems. In this approach, the baseband signal processing algorithms are broken into smaller tasks undertaken by high performance dedicated processing units or hardware accelerators. The hardware accelerators are flexible hardware structures which are reconfigurable and scalable.

This dissertation explored dynamically configurable VLSI architectures for implementation of MIMO detection algorithms. The main objective in this study was to introduce architectures with the following main characteristics:

Throughput: The main idea behind using MIMO is to increase the overall data throughput of the communication link. MIMO detection algorithms are highly complex algorithms. Processing latency of detector is quite important as it directly affects throughput of the system.

Configurability: The upcoming wireless standards support multiple operating modes for transceivers. Configurability becomes an increasingly important performance measure in VLSI design. We proposed new algorithms and architectures to minimize the cost and impact of configurability to the design.

Area/Power: Silicon area is one of the main design factors for battery operated and mobile devices as it directly affects manufacturing costs, yield and consumed power.

In this thesis, we considered tree search based algorithms as the basis for detector design. Tree search based algorithms can be divided into two main groups, i.e. depth-first and breadth-first search algorithms. As the first step, we proposed reference configurable multi-mode designs for future comparisons. Later, we proposed architectures based on a configurable processing element which has been called MECU (Metric Enumeration and Computation Unit). It has been shown that MECU block can be used in folding and pipelined architectures to implement tree search based detection algorithms.

In case of depth-first sphere decoding algorithm, a low complexity solution has been proposed to support BPSK, QPSK and 16-QAM modulation schemes as well as different antenna configurations. A number of modifications to the original depth-first sphere decoding algorithm have been proposed to improve performance of the depth-first sphere decoding algorithm. It has been shown that Depth-First-K-Best (DFKB) algorithm with level based radius definition can be used to support 64-QAM modulation with reduced complexity. The key design challenge for implementation of DFKB is a low complexity enumeration algorithm that can find and sort the lowest K out of 64 metrics at each node. We proposed an enumeration algorithm with low hardware complexity for enumeration of K out of 64 possible child nodes. MECU block which has been designed based on this enumeration algorithm is a configurable block that supports BPSK, QPSK, 16-QAM and 64-QAM modulation modes. MECU has been used to introduce new architectures for depth-first sphere decoding, fixed sphere decoding and K-best algorithms. Implementation results show that area and throughput of detector designs based on

MECU processing elements have been improved significantly compared to reference designs.

As proof of concepts, an emulation platform has been designed and implemented based on hardware-in-the loop (HIL) technology to test and optimize the performance of proposed architectures.

7.2 Future work

The receiver detection error rate performance can be improved by applying iterative soft-MIMO detectors. The architectures proposed in this dissertation can be used to provide the soft information needed in an iterative MIMO detector. In hard-output detection, the search tree has been searched to find the symbol vector that minimizes $\|\mathbf{y} - \mathbf{Hs}\|^2$. In the soft detection algorithms such as List Sphere Decoding (LSD) algorithm (section 3.5.1) a list of the N_{cand} points that makes $\|\mathbf{y} - \mathbf{Hs}\|^2$ smallest are generated. The architectures proposed in this dissertation for hard detection can be modified to search for the candidate symbol list and soft-values computation of APPs (Appendix A).

The algorithmic exploration in this dissertation shows that different algorithms have limited range of effectiveness in terms of throughput and BER performance. A Flexible MIMO detector architecture that can respond to channel variations and reconfigure itself and implement different detection algorithms can provide optimum performance under different operating conditions. The approach in this dissertation was to implement tree search based algorithm by using *MECU* processing element. The next logical step would be to design a flexible architecture that can reconfigure connections between *MECU* and buffer blocks such that implementation of different detection algorithms is possible within one general detector architecture.

Appendix A: APP Detection

In the following derivation of the basic equations for APP detection, it is assumed that the logical zero for a bit is represented by x_k=-1 and logical one by x_k=+1. In this section we briefly review the basic equations required in APP detection; full description and proof of equations can be found in [7][8][48]. The a-posteriori L-value of the bit x_k, k = 0,...,M.M_c-1 (M=N_t, M_c=Q), conditioned on the received vector channel **y**, is defined [47] as

$$L_D(x_k \mid \mathbf{y}) = \ln \frac{P[x_k = +1 \mid \mathbf{y}]}{P[x_k = -1 \mid \mathbf{y}]} \tag{A-1}$$

We can make the assumption that the bits in **x** have been encoded with a channel code, but that an interleaver at the encoder is used to scramble the bits from other blocks, so that the bits within **x** are statistically independent of one another. Using Bayes' theorem and knowing $x_0,\ldots,x_{M.Mc-1}$ are statistically independent, we can write soft output values as

$$L_D(x_k \mid \mathbf{y}) = L_A(x_x) + \ln \frac{\sum_{x \in X_{k,+1}} p(\mathbf{y} \mid x) \exp \sum_{j \in J_{k,x}} L_A(x_j)}{\sum_{x \in X_{k,-1}} p(\mathbf{y} \mid x) \exp \sum_{j \in J_{k,x}} L_A(x_j)} \tag{A-2}$$

And $L_E(x_k \mid \mathbf{y})$ is defined as

$$L_E(x_k \mid \mathbf{y}) = \ln \frac{\sum_{x \in X_{k,+1}} p(\mathbf{y} \mid x) \exp \sum_{j \in J_{k,x}} L_A(x_j)}{\sum_{x \in X_{k,-1}} p(\mathbf{y} \mid x) \exp \sum_{j \in J_{k,x}} L_A(x_j)} \tag{A-3}$$

where $X_{k,+1}$ is the set of $2^{M.Mc-1}$ bit vectors **x** having x_k = +1 i.e.

$$X_{k,+1} = \{\mathbf{x} \mid x_k = +1\} \tag{A-4}$$

$$X_{k,-1} = \{\mathbf{x} \mid x_k = -1\} \tag{A-5}$$

And $J_{k,x}$ is the set of indices j with

$$J_{k,x} = \{j \mid j = 0,\ldots,M.M_c - 1, j \neq k, x_j = 1\} \tag{A-6}$$

$$L_A(x_j) = \ln \frac{P[x_j = +1]}{P[x_j = -1]} \tag{A-7}$$

It is shown in [7] and [8] that (A-2) and (A-3) can be further simplified as

$$L_D(x_k \mid \mathbf{y}) = L_A(x_x) + \ln \frac{\sum_{x \in X_{k,+1}} p(\mathbf{y} \mid x) \exp\left(\frac{1}{2} \mathbf{x}_{[k]}^T \cdot \mathbf{L}_{A,[k]}\right)}{\sum_{x \in X_{k,-1}} p(\mathbf{y} \mid x) \exp\left(\frac{1}{2} \mathbf{x}_{[k]}^T \cdot \mathbf{L}_{A,[k]}\right)} \tag{A-8}$$

$$L_E(x_k \mid \mathbf{y}) = \ln \frac{\sum_{x \in X_{k,+1}} p(\mathbf{y} \mid x) \exp\left(\frac{1}{2} \mathbf{x}_{[k]}^T \cdot \mathbf{L}_{A,[k]}\right)}{\sum_{x \in X_{k,-1}} p(\mathbf{y} \mid x) \exp\left(\frac{1}{2} \mathbf{x}_{[k]}^T \cdot \mathbf{L}_{A,[k]}\right)} \tag{A-9}$$

In (A-8), $\mathbf{x}_{[k]}$ denotes the subvector of $\mathbf{x}$ obtained by omitting its kth element x_k, and $\mathbf{L}_{A,[k]}$ denotes the vector of all L_A values, also omitting x_k. Thus we can see that L_D can be written as a sum of *a priori* L-value L_A and extrinsic L-value L_E. Based on the subscripts used in Figure 3.11 we have

$$L_{D1}(x_{1,k} \mid \mathbf{y}) = L_{A1}(x_{1,x}) + \ln \frac{\sum_{x_1 \in X_{k,+1}} p(\mathbf{y} \mid x_1) \exp\left(\frac{1}{2} \mathbf{x}_{1,[k]}^T \cdot \mathbf{L}_{A1,[k]}\right)}{\sum_{x_1 \in X_{k,-1}} p(\mathbf{y} \mid x_1) \exp\left(\frac{1}{2} \mathbf{x}_{1,[k]}^T \cdot \mathbf{L}_{A1,[k]}\right)} \tag{A-10}$$

$$L_{E1}(x_{1,k} \mid \mathbf{y}) = \ln \frac{\sum_{x_1 \in X_{k,+1}} p(\mathbf{y} \mid x_1) \exp\left(\frac{1}{2} \mathbf{x}_{1,[k]}^T \cdot \mathbf{L}_{A1,[k]}\right)}{\sum_{x_1 \in X_{k,-1}} p(\mathbf{y} \mid x_1) \exp\left(\frac{1}{2} \mathbf{x}_{1,[k]}^T \cdot \mathbf{L}_{A1,[k]}\right)} \tag{A-11}$$

Equation (A-1) can also be applied to the channel error-correcting code and we will have the a-posteriori L-value obtained from APP decoding of the outer channel code. Thus the channel decoder processing can be decomposed into a priori and extrinsic components.

$$L_{D2}(x_{2,k} \mid \mathbf{L}_{A2}) = L_{A2}(x_{2,x}) + \ln \frac{\sum_{x_2 \in X_{k,+1}} \exp\left(\frac{1}{2} \mathbf{x}_{2,[k]}^T \cdot \mathbf{L}_{A2,[k]}\right)}{\sum_{x_2 \in X_{k,-1}} \exp\left(\frac{1}{2} \mathbf{x}_{2,[k]}^T \cdot \mathbf{L}_{A2,[k]}\right)} \tag{A-12}$$

$$L_{E2}(x_{2,k} \mid \mathbf{L}_{A2,[k]}) = \ln \frac{\sum_{x_2 \in X_{k,+1}} \exp\left(\frac{1}{2}\mathbf{x}_{2,[k]}^T \cdot \mathbf{L}_{A2,[k]}\right)}{\sum_{x_2 \in X_{k,-1}} \exp\left(\frac{1}{2}\mathbf{x}_{2,[k]}^T \cdot \mathbf{L}_{A2,[k]}\right)} \quad \textbf{(A-13)}$$

In this equation the raw data bits are denoted by $\mathbf{x_2}$.

Likelihood function for APP detection

The likelihood function $p(\mathbf{y} \mid \mathbf{x})$ can be calculated using the channel model [7].

$$p(\mathbf{y} \mid s = map(x)) = \frac{\exp\left[-\frac{1}{2\sigma^2} \cdot \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2\right]}{(2\pi\sigma^2)^N} \quad \textbf{(A-14)}$$

Implementation Simplifications

We can use the *Jacobian Logarithm* to further simplify equation (A-11).

$$\text{jacln}(a_1, a_2) := \ln\left(e^{a_1} + e^{a_2}\right) = \max(a_1 + a_2) + \ln\left(1 + e^{-|a_1 - a_2|}\right) \quad \textbf{(A-15)}$$

$$r(|a_1 - a_2|) \approx \ln\left(1 + e^{-|a_1 - a_2|}\right) \quad \textbf{(A-16)}$$

Where r(.) can be viewed as a refinement of the coarse approximation $\max(a_1 + a_2)$.

In a DSP implementation, a Jac-log approximation can be obtained by storing r(.) in a look-up table [7] [9]. With the max-log approximation, the extrinsic L-value of (A-11) can be approximated as

$$L_E(x_k \mid \mathbf{y}) \approx \frac{1}{2} \max_{x \in X_{k,+1}} \left\{-\frac{1}{\sigma^2} \cdot \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 + \mathbf{x}_{[k]}^T \cdot \mathbf{L}_{A,[k]}\right\} - \frac{1}{2} \max_{x \in X_{k,-1}} \left\{-\frac{1}{\sigma^2} \cdot \|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2 + \mathbf{x}_{[k]}^T \cdot \mathbf{L}_{A,[k]}\right\}$$

(A-17)

where s=map(x).

References

[1] U. Fincke and M. Phost, "Improved methods for calculating vectors for short length in a lattice, including a complexity analysis," *Math. Computer*, vol. 44, pp. 463-471, April 1985.

[2] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications," in *Proc. ACM SIGSAM*, 1981, pp. 37–44.

[3] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Trans. Inform. Theory*, vol. 48, no. 8, pp. 2201-2214, Aug. 2002.

[4] C. P. Schnorr and M. Euchner, "Lattice basis reduction: Improving practical lattice basis reduction and solving subset sum problems", *Math. Programming*, vol. 66, pp. 181-191, 1994.

[5] M. O. Damen, H. El Gamal, and G. Caire, "On maximum likelihood detection and the search for the closest lattice point," *IEEE Trans. Information Theory,* vol. 49, No. 10, pp. 2389-2402, Oct. 2003.

[6] M. O. Damen, A. Chkief, and J. C. Belfiore, "Lattice code decoder for space-time codes," *IEEE Communications letters,* vol. 4, No. 5, pp. 161-163, May 2000.

[7] B. M. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Trans. Commun.*, vol. 51, pp. 389-399, Mar. 2003.

[8] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary and block convolutional codes," *IEEE Trans. Inform. Theory,* vol. 42, pp. 429-445, Mar. 1996.

[9] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain," in *Proc. Int. Conf. Communications,* June 1995, pp. 1009-1013.

[10] J. Hagenauer, P. Robertson, and L. Papke, "Iterative (turbo) decoding of systematic convolutional codes with the MAP and SOVA algorithms," in *Proc. ITG Symp. Source and Channel Coding,* 1994, pp. 21-29.

[11] S. Baro, J. Hagenauer, and M. Witzke, "Iterative detection of MIMO transmission using a list-sequential (LISS) detector," *Proc. Of IEEE International Conference on Communications,* May 2003, pp. 2653-2657.

[12] K. Wong, C. Tsui, R. Cheng, and W. Mow, "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," in *Proc. IEEE Int. Symp. On Circuits and Systems (ISCAS)*, vol. 3, May 2002, pp. 273-276.

[13] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detectors using the sphere decoding algorithm", *IEEE Journal of Solid-State Circuits*, vol. 40. Issue 7, pp. 1566-1577, July 2005.

[14] D. Garrett, L. Davis, S. ten Brink, B. Hochwald, and G. Knagge, "Silicon complexity for maximum likelihood MIMO detection using spherical decoding," *IEEE journal of Solid-State Circuits,* vol.39, pp. 1544-1552, Sept. 2004.

[15] Z. Guo and P. Nilsson, "A VLSI architecture for the Schnorr-Euchner decoder for MIMO systems," *Proc. IEEE CAS Symposium on Emerging Technologies,* June 2004, pp. 65-68.

[16] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-Best sphere decoding for MIMO detection", *IEEE Journal on Selected Areas in Communications*, vol. 24, no.3, pp. 491-503, Mar. 2006.

[17] S. Seo and S. Park, "Efficient VLSI implementation of the List Sphere Decoder with real-value based tree searching method", *Proc. IEEE ICACT*, Feb. 2006, pp. 1694-1697.

[18] Z. Guo and P. Nilsson, "VLSI implementation issues of lattice decoders for MIMO systems," *Proc. IEEE Int. Symp. On circuits and systems (ISCAS),* vol.4 May 2004, pp. 477-480.

[19] Z. Guo and P. Nilsson, "A 53.3 Mb/s 4x4 16-QAM MIMO decoder in 0.35 μm CMOS," *Proc. IEEE Int. Symp. On circuits and systems (ISCAS),* May 2005, pp. 4947-4950.

[20] Z. Guo and P. Nilsson, "A VLSI implementation of MIMO detection for future wireless communications," *Proc. IEEE PIMRC'03*, vol. 3, 2003, pp. 2852–2856.

[21] D. Garrett, G. Woodward, L. Davis, G. Knagge, and C. Nicol, "A 28.8 Mb/s 4x4 MIMO 3G high-speed downlink packet access receiver with normalized least mean square equalization," *IEEE ISSCC Dig. Tech. Papers*, vol. 1, Feb. 2004, pp. 420.

[22] A. Burg, N. Felber, and W. Fichtner, "A 50 Mbps 4 x 4 maximum likelihood decoder for multiple-input multiple-output systems with QPSK modulation," *Proc. IEEE Int. Conf. Electron.*, Circuits, System (ICECS), vol. 1, 2003, pp. 332–335.

[23] A. Burg, M. Wenk, M. Zellweger, M. Wegmueller, N. Felber, and W. Fichtner, "VLSI implementation of the sphere decoding algorithm," *Proc. ESSCIRC 2004*, Leuven, Belgium, Sep. 2004, pp. 303–306.

[24] A. Burg, M. Borgmann, C. Simon, M. Wenk, M. Zellweger, and W. Fichtner, "Performance tradeoffs in the VLSI implementation of the sphere decoding algorithm," *Proc. IEE 3G Mobile Communication Conf.*, London, U.K., Oct. 2004, pp. 93–97.

[25] URL: http://www.altera.com/products/devices/stratix-fpgas/stratix-ii/

[26] D. Garrett, L. Davis, G. Woodward, "19.2 Mbits/s 4 x 4 BLAST/MIMO detector with soft ML outputs", *IEE Electron. Letter*, VOL 39, NO. 2, Jan. 23, 2003, pp. 233-235.

[27] A. Burg, M. Borgmann, M. Wenk, C. Studer, H. Bölcskei, "Advanced Receiver Algorithms for MIMO Wireless Communications", *Proc. Design Automation and Test in Europe Conference, 2006*, pp. 593-598.

[28] A. Burg, S. Haene, D. Perels, P. Luethi, N. Felber, and W. Fichtner, "Algorithm and VLSI Architecture for Linear MMSE Detection in MIMO-OFDM Systems," *Proc. of IEEE Int. Symp. on Circuits and Systems (ISCAS)2006*, 2006, pp. 4102-4105.

[29] M. S. Gast, *802.11 Wireless Networks: The Definitive Guide*, O'Reilly, 2nd Edition, April 2005.

[30] J. Lorincz, D. Begusic, "Physical layer analysis of emerging IEEE 802.11n WLAN standard," *Proc. Advanced Communication Technology ICACT 2006*, volume 1, pp. 189-194.

[31] URL: http://www.ieee802.org/

[32] URL: http://www.3gpp.org/

[33] N. Voros and K. Masselos, *System Level Design of Reconfigurable Systems-on-Chip*, Springer, 2005.

[34] Y. L. de Jong, T. J. Willink, "Iterative tree search detection for MIMO wireless systems," in *Proc. IEEE 56th Vehicular Technology Conference*, Sep. 2002, pp. 1041-1045.

[35] J. Jie, C.-Y. Tsui, and W.-H. Mow, "A threshold-based algorithm and VLSI implementation of a K-best lattice decoder for MIMO systems," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)2005*, 2005, pp. 3359-3362.

[36] M. Wenk, M. Zellweger, A. Burg, N. Felber, and W. Fichtner, "K-Best MIMO detection VLSI architectures achieving up to 424 Mbps," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)2006*, 2006, pp. 1151-1154.

[37] A. J. Paulraj, D. A. Gore, R. U. Nabar, H. Bolcskei, "An Overview of MIMO. Communications: A Key to Gigabit Wireless", in *Proc. of IEEE*, Vol. 92, No.2, Feb. 2004, pp.198-218.

[38] H. Bolcskei, "MIMO-OFDM Wireless Systems: Basics, Perspectives, and Challenges," in *IEEE Wireless Communications*, Vol. 13, No. 4, Aug. 2006, pp. 31-37.

[39] A. J. Paulraj, R. U. Nabar, and D. A. Gore, *Introduction to Space-Time Wireless Communications*, Cambridge, UK: Cambridge Univ. Press, 2003.

[40] A. Chandrakasan, R. Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 4th edition, 2000.

[41] J. Wang, B. Daneshrad, "A Comparative Study of MIMO Detection Algorithms for Wideband Spatial Multiplexing Systems," in *Proceedings: 2005 IEEE Wireless Communication and Networking Conference, New Orleans, Louisiana*, March 2005, Vol. 1, pp. 408-413.

[42] D. Perels, S. Hane, P. Luethi, A. Burg, N. Felber, W. Fichtner, and H. Bolcskei, "ASIC implementation of a MIMO-OFDM transceiver for 192 Mbps WLANs," in *European Solid-State Circuits Conference (ESSCIRC)*, Sept. 2005, pp. 215–218.

[43] G. Estrin, B. Bussell, R. Turn, and J. Bibb, "Parallel Processing in a restructurable Computer System", IEEE Transactions on Electronic Computers, 1963, pp. 747-755.

[44] URL: http://www.xilinx.com/products/virtex6/

[45] P. Wolniansky, G. Foschini, G. Golden, and R. Valenzuela, "VBLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel," in Proc. *IEEE Int. Symp. On Signals, Systems, and Electronics (ISSSE)*, Oct. 1998, pp. 295-300.

[46] H. Jafarkhani, *Space-Time Coding: Theory and Practice*, Cambridge University Press, 2005.

[47] H. Bolcskei, D. Gesbert, C. B. Papadias, and A.-J. van der Veen, *Space-Time Wireless Systems: From Array Processing to MIMO Communications*, Cambridge University Press, 2006.

[48] H. Vikalo, B. Hassibi, and T. Kailath, "Iterative decoding for MIMO channels via modified sphere decoder" in *IEEE Trans. Wireless Communication*, Nov. 2004, Vol.3, No.6, pp. 2299-2311.

[49] R. Shariat-Yazdi and T. Kwasniewski, "A high performance VLSI architecture for MIMO detection in future WLAN receivers", in Proc. *Canadian Conference of Electrical and Computer Engineering (CCECE)*, April 2007.

[50] R. Shariat-Yazdi and T. Kwasniewski, "Challenges in the design of next generation WLAN terminals", in Proc. *Canadian Conference of Electrical and Computer Engineering (CCECE)*, April 2007.

[51] P. Radosavljevic and J. R. Cavallaro, "Soft sphere detection with bounded search for high-throughput MIMO receivers," in Proc. *40th annual Asilomar Conference on Signals, Systems, and Computers*, November 2006.

[52] S. Mathew and R. Krishnamurthy, "Circuit Technologies for High-performance Energy-efficient Sub-45nm Microprocessors", in *2006 Emerging Technologies Workshop*, July 19-21 Banff Canada.

[53] K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation,* Wiley-Interscience, 1999.

[54] D. Markovic, B. Nikolic, and R. W. Brodersen, "Power and area efficient VLSI architectures for communication signal processing," in Proc. *IEEE International Conference on Communications ICC'06*, Jun 2006, Vol. 7, pp. 3223-3228.

[55] K. E. Batcher, "Sorting networks and their applications," *Proc. AFIPS Spring Joint Computer Conference,* 1968, pp. 307-314.

[56] URL: http://www.altera.com/products/devices/cyclone2/cy2-index.jsp

[57] URL: http://www.altera.com/products/devices/stratix-fpgas/stratix-iii/st3-index.jsp

[58] R. Shariat-Yazdi and T. Kwasniewski, "A Multi-mode Sphere Detector Architecture for WLAN Applications", in Proc. IEEE System on Chip Conference (SOCC08), Sept. 2008.

[59] R. Shariat-Yazdi and T. Kwasniewski, "Low complexity sphere decoding algorithms", in Proc. IEEE International Symposium on Wireless Communication Systems (ISWCS08), Oct. 2008.

[60] M. Shafi *et al.*, "Special Issues on MIMO Systems and Applications (I/II)", IEEE Journal on Selected Areas in Communications, Vol. 21, April/June 2003.

[61] R. Shariat-Yazdi and T. Kwasniewski, "Reconfigurable K-best MIMO detector architecture and FPGA implementation", Proc. International Symposium on Intelligent Signal Processing and Communication Systems, ISPACS 2007, November 2007.

[62] R. Shariat-Yazdi and T. Kwasniewski, "Dual Mode K-Best MIMO Detector Architecture and VLSI Implementation", Proc. 14th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2007, December 2007.

[63] R. Shariat-Yazdi and T. Kwasniewski, "Configurable K-best MIMO Detector Architecture", Proc. 3rd International Symposium on Communications, Control and Signal Processing, ISCCSP 2008, March 2008.

[64] R. Shariat-Yazdi and T. Kwasniewski, "Low Complexity VLSI Architecture for MIMO Sphere Decoding Algorithm" , Proc. 2009 IEEE NEWCAS-TAISA Conference.

[65] S. Chen, T. Zhang, and Y. Xin, "Relaxed K-best MIMO Signal Detector Design and VLSI Implementation", IEEE Trans. VLSI Systems, Vol. 15, No.3, March 2007, pp. 328-337.

[66] H. Lin, R. Chang, and H. Chen, "A High-Speed SDM-MIMO Decoder Using Efficient Candidate Searching for Wireless Communication", IEEE Trans. Circuit and Systems II, Vol. 55, No.3, March 2008, pp. 289-293.

[67] M. Gokhale and P. Graham, *Reconfigurable Computing: Accelerating Computation With Filed Programmable Gate Arrays*, Springer, 2005.

[68] S. Srikanteswara, R. Palat, J. Reed, and P. Athanas, "An Overview of Configurable Computing Machines for Software Radio Handsets", IEEE Communication Magazine, July 2003, pp. 134-141.

[69] G. Rauwerda, P. Heysters, and G. Smit, "Towards Software Defined Radios Using Coarse-Grained Reconfigurable Hardware", IEEE Trans. Very Large Scale Integration (VLSI) Systems, Vol. 16, No. 1, January 2008, pp. 3-13.

[70] C. Huang, C. Yu, and H. Ma, "A Power-Efficient Configurable Low-Complexity MIMO Detector", IEEE Trans. Circuits and Systems I, Vol. 56, No. 2, February 2009, pp. 485-496.

[71] M. Shabany and P. Gulak, "A 0.13 μm CMOS 655 Mbs/s 4x4 64-QAM K-Best MIMO Detector", in Proc. 2009 IEEE International Solid State Conference (ISSCC 2009), February 2009.

[72] B. Hassibi and H. Vikalo, "On the Sphere-Decoding Algorithm I. Expected Complexity", IEEE Trans. Signal Processing, Vol. 53, No. 8, August 2005, pp. 2806-2818.

[73] L. Barbero and J. Thompson, "Fixing the Complexity of the Sphere Decoder for MIMO Detection", IEEE Trans. Wireless Communications, Vol. 7, No. 6, June 2008, pp. 2131-2142.

[74] B. Widrow and S. Stearns, *Adaptive Signal Processing,* Prentice-Hall, 1985.

[75] S. Hauck and A. Dehon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*, Elsevier, 2008.

[76] D. Lattard *et al.*, "A Reconfigurable Baseband Platform Based on an Asynchronous Network-on-Chip", IEEE Journal of Solid-State Circuits, Vol. 3, No. 43, January 2008, pp. 223-235.

[77] M. Rupp, G. Gritsch, and H. Weinrichter, "Approximate ML detection for MIMO systems with very low complexity," in Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vol. 4, May 2004.